Systems
Optimization
Laboratory

Dynamic Pricing Criteria in Linear Programming

by
Edward S. Klotz

TECHNICAL REPORT SOL 88-15

July 1988

Department of Operations Research
Stanford University
Stanford, CA 94305

2

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

Dynamic Pricing Criteria in Linear Programming

by
Edward S. Klotz

TECHNICAL REPORT SOL 88-15

July 1988

Research and reproduction of this report were partially supported by the National Science Foundation Grants DMS-8420623, SES-8518662, ECS83-12142; U.S. Department of Energy Grant DE-FG03-87ER25028, and Office of Naval Research Contract N00014-85-K-0343.

DYNAMIC PRICING CRITERIA IN LINEAR PROGRAMMING

Edward S. Klotz
Department of Operations Research, Stanford University

In recent years the interest in linear programming algorithms has increased greatly due to the discovery of new interior-point methods. New results have also prompted researchers to reconsider some previously discarded ideas in light of their additional knowledge. This thesis begins with a study of some variants of the reduced-gradient method applied to linear programs. Preliminary computational tests revealed how sparsity and degeneracy, two characteristics present in most practical problems, can severely inhibit such variants. The development of dynamic pricing criteria to exclude certain columns from the search direction provides a computationally efficient way to alleviate these difficulties. Application to the simplex method yields a pivot rule designed to avoid degenerate pivots. Generalization of the rule yields a cheap method to estimate the step lengths associated with potential incoming nonbasic variables. The result is a set of pivot rules that appear particularly useful on highly degenerate and poorly scaled linear programs. Extensive computational tests are presented.

Key words: linear programming, simplex method, reduced-gradient method, pricing, degeneracy.

ACKNOWLEDGEMENTS

It is probably impossible to acknowledge all of the people who somehow influenced the writing of this work, but I shall try to do so. I apologize to anyone I may have omitted.

Let me begin with those who contributed most directly. First of all, I wish to thank Professor George Dantzig, my thesis advisor. His experience, knowledge, and intuition provided uncountable contributions to this work, and he helped make the educational process an interesting and enjoyable experience. I also appreciate the availability of his time, a truly scarce resource for such a well-known individual.

I also wish to thank Professors Michael Saunders and Curtis Eaves, the other two members of my reading committee. Their contributions greatly exceeded their constructive insights on the preliminary drafts. Michael Saunders is one of the creators of MINOS, a linear and nonlinear optimization code that was vital to many of the experiments chronicled in this document. Without it, the time required to properly implement many of the ideas developed here would have increased greatly. Curtis Eaves directed many of the seminars that strongly influenced and improved my understanding of linear programming. Those courses established a solid foundation for the research described herein.

In addition, I wish to thank Professor Richard Cottle, the fourth member of my orals committee, and Professor Michael Todd of Cornell University, who supplied some constructive feedback on the first draft of this work.

Professors alone do not create an enjoyable educational experience. The majority of one's time is spent with fellow students. I wish to thank all of the students in the Department of Operations Research at Stanford for helping to create a healthy learning environment. I especially appreciate Irv Lustig, Karel Zikan and Sam Eldersveld for many enlightening conversations about optimization that have improved this document.

Thus far I have only mentioned individuals I met while at Stanford. Many others made great contributions by educating me to the level where I could benefit from the opportunity to study here. I suspect that all of my teachers made positive contributions, but two in particular emerge in my mind as having played essential roles without which I would not have started, let alone finished, this work. I wish to thank Richard Murphy, who taught me mathematics during my sophomore and junior years at New Trier East High School in Winnetka, Illinois. The improvement I experienced under his tutelege was indeed astounding, and I feel very fortunate to have received such good teaching. Second, I wish to thank Dr. Samuel Goldberg, who instructed me in four different courses during my undergraduate years at Oberlin College, including my first one in operations research. He is a truly outstanding teacher.

Naturally, I also wish to thank all of the members of my family for their encouragement. In particular, however, I thank Irving Klotz, my father. As a professor of chemistry at Northwestern University, his own experiences enabled him to pro-

vide tremendous support during my stay at Stanford, especially during the difficult times.

Finally, I wish to thank the following random assortment of people for improving the quality of life during my time here:

Ed Brown,
Jacqui Lewis,
Dana Perry,
Scott Morris,
Steve Shapiro,
Sam Eldersveld,
Irv Lustig,
Lila Noonkester,
Peanut Harms,
Terence Boynton,
Dirk Rohloff,
the rest of the Thursday Night Brawlers,
the O.R. Jazz Band,
Mr. Green and the Men from Modesto,
Mark Perkins,
Fred Krueger,
Bill Peterson and
anyone else I forgot to mention.

Stanford University,
May, 1988.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

After more than 40 years, Dantzig's simplex algorithm remains the most popular method for solving linear programs. Since its invention in 1947, researchers have proposed many other algorithms, yet none of these has consistently outperformed the simplex algorithm. In addition, many different pivot rules (see [2], [17], [18], [19], [43], [52] and [53]) have arisen for the method, yet most implementations continue to use Dantzig's original pivot rule.

In 1984 Karmarkar [23] proposed a projective algorithm for linear programming substantially different from the simplex method. Karmarkar's algorithm moves through the interior of the feasible region, while the simplex method traverses a sequence of feasible vertices. While it remains unclear if this new algorithm will ultimately replace the simplex method for solving practical problems (refer to [1] and [26]), the approach has profoundly influenced the direction of research in linear programming. Other algorithms have arisen (see [4], [42], [49] and [54]) that are strongly motivated by the projective method. In addition, it has prompted researchers to reexamine previously discarded ideas in light of their new knowledge (see [15]). The research described in this thesis began by adopting the latter approach, experimenting with variants of the reduced-gradient method applied to linear programming. Although these experiments did not yield an algorithm superior to Dantzig's method, they provided insight into the major drawbacks of these variants. The experiments also inspired a set of dynamic pricing criteria that shows great promise to improve the simplex method.

Chapters 2, 3 and 4 comprise the majority of the thesis. Chapter 2 discusses a set of feasible direction methods for linear programming similar to the reduced-gradient method. After a brief description, we examine how degeneracy can inhibit the progress of such algorithms. This obstacle motivates the development of pricing criteria to avoid degeneracy. The idea also generalizes to deal with "near" degeneracy. The result is a modified feasible direction method. We present computational results for the method on some practical problems. The algorithm usually outperformed the simplex method with respect to iterations, but it always required more time. Nonetheless, the results reveal an improvement over previous tests of similar methods. Pricing criteria to avoid degeneracy provide the main source of improvement.

Chapter 3 considers the use of such pricing criteria in the context of the simplex method. We first apply the results of Chapter 2. We then develop additional criteria designed specifically to improve the simplex method. The result is a set of very promising multiple-priority pivot rules. One can view such procedures as computationally inexpensive attempts to estimate the step length associated with a potential basic variable. Although fairly cheap, these techniques do increase the computational effort. We therefore develop results designed to reduce the extra work. In addition, we study a parametric simplex method due to Gass and Saaty that Dantzig recommends in [7] to avoid cycling. Although this approach usually

performs well on highly degenerate problems, it does so without making any effort to avoid degenerate pivots. This leads to a pivoting procedure that combines the parametric method's selection rule with a multiple-priority pivot rule designed to avoid degenerate pivots. The chapter concludes with some results designed to reduce the computational burden associated with the previously described rules.

Chapter 4 presents extensive computational tests of the pivot rules described. Each rule requires fewer iterations than the simplex method on most problems. However, the reduction in iterations does not always result in a reduction in computation time. Nonetheless, the rules typically perform quite well on the larger, more difficult problems.

Chapter 5 contains a discussion of some untested ideas that may be fruitful areas for future research. Extensions of the ideas developed here, along with applications to nonlinear programming, are considered.

Chapter 6 summarizes the thesis and provides some conclusions. The author hopes that the reader will have acquired some additional tools in his arsenal for solving linear programs, as well as some new insight on some previously established results.

An appendix follows the thesis. It describes details pertinent to the computational experiments; for example, the ideas described in Chapters 2 and 3 had to be generalized to deal with bounded variables. The appendix contains no new mathematics, but it highlights significant differences between the theory and practice of linear programming.

1.1. Notation

This thesis deals primarily with linear programming. We begin by establishing some pertinent preliminaries and notation. The author assumes that the reader is familiar with standard linear programming theory; if not, refer to [6] or [35]. For details on the computational aspects of linear programming, see [14], [16], [31] and [46]. We consider the following standard form linear programming problem:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0, \end{aligned} \tag{1.1}$$

where $A \in R^{m \times n}$, $c \in R^n$, $x \in R^n$, and $b \in R^m$. Without loss of generality, assume that $m \leq n$ and A has full rank.

The dual of (1.1) is:

$$\begin{aligned} \text{maximize} \quad & b^T \pi \\ \text{subject to} \quad & A^T \pi + v = c \\ & v \geq 0. \end{aligned} \tag{1.2}$$

We will often wish to consider submatrices of A. Consider $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$ and $J = \{j_1, \ldots, j_l\} \subseteq \{1, \ldots, n\}$. Then

$$A_{I}$$

represents the submatrix of A consisting of rows indexed by $\{i_1, \ldots, i_k\}$ and all columns. Similarly,

$$A_{\cdot J}$$

represents the submatrix of A consisting of columns indexed by $\{j_1, \ldots, j_l\}$ and all rows. Also,

$$A_{I,J}$$

consists of the submatrix of A with rows indexed by I and columns indexed by J. This notation simplifies slightly for vectors; c_J denotes the components of c indexed by J.

Let B be an $m \times m$ nonsingular submatrix of A. B is called a basis. Such a B exists because A has full rank. When appropriate, we shall also use $B = \{j_1, \ldots, j_m\}$ to index the corresponding columns of A; thus B and $A_{\cdot B}$ represent the same matrix. Similarly, let N index the set of nonbasic columns of A. Given a basis B, the linear program (1.1) has the following equivalent canonical form:

$$\begin{aligned} \text{minimize} \quad & \bar{c}_N^T x_N \\ \text{subject to} \quad & x_B + \bar{A}_{\cdot N} x_N = \bar{b} \\ & x_B, x_N \geq 0 \end{aligned} \tag{1.3}$$

where $\bar{A} = B^{-1}A$, $\bar{b} = B^{-1}b$, and $\bar{c}_N^T = c_N^T - \pi^T A_{\cdot N}$. The m-vector π comprises the basic dual variables and is defined by the linear system

$$\pi^T B = c_B^T.$$

The linear programs (1.1) and (1.3) provide the framework for the ideas developed herein. Additional notation shall be defined as the need arises.

CHAPTER 2: FEASIBLE DIRECTION METHODS

2.1. Preliminaries

Wolfe proposed the reduced-gradient method (see [50]) for linearly constrained optimization in 1962. While primarily intended for nonlinear objective functions, the algorithm can also solve linear programs. Since its invention, several authors (see [5], [8], [9], [10], [12], [21], [22], [36], [37], [40] and [45]) have described similar algorithms for linear programs. One can view these approaches as extensions of the simplex method since they utilize the notions of basic and nonbasic variables yet allow for feasible iterates other than vertices. Computational results in the literature usually deal only with random problems; one exception consists of tests performed by Kallio and Orchard-Hays [22]. The purpose of this chapter is to broaden our understanding of such algorithms. We begin with a general description of the approach, provided in Sections 2.2–2.4. Section 2.5 discusses the effects of degeneracy on such algorithms, while Section 2.6 develops techniques to deal with this problem. Section 2.7 then proposes a modified feasible direction method based on the results of Sections 2.2–2.6. Computational results on a set of practical test problems are presented in Section 2.8.

2.2. The Choice of Search Direction

Consider the linear programs (1.1) and (1.3). Assume a feasible solution x and a basis B. Unlike in the simplex method, x here need not be a vertex, and B need not be a feasible basis. We wish to choose a search direction $q \in R^n$ such that $Aq = 0$ and $c^T q < 0$. Choose a set $P \subseteq N$ of promising variables to change. For $j \in P$, set δ_j as the rate of change of x_j; let $\delta_j = 0$ for $j \notin P$. We will elaborate on how to determine δ_j and P later. Now, define the following search direction q:

$$\begin{aligned} q_P &= \delta_P \\ q_{N \setminus P} &= 0 \\ q_B &= -\overline{A}_{\cdot P} q_P. \end{aligned} \tag{2.1}$$

Note that $Aq = Bq_B + A_{\cdot P} q_P + A_{\cdot N \setminus P} q_{N \setminus P} = 0$. For any suboptimal x we will see in Section 2.3 that there always exists a set P and rate of change δ_P such that $c^T q < 0$. Therefore, q provides a descent direction. Observe that if B is a feasible basis and x is the associated vertex, then one can view the simplex method as a special case by choosing a single promising variable with negative reduced cost $\bar{c}_j$ and setting $\delta_j = 1$.

Given q, we wish to move to a new solution

$$\overline{x} = x + \theta q \tag{2.2}$$

for any nonnegative scalar θ. Since q lies in the null space of A, it follows that $A\overline{x} = b$, so we wish to choose θ to maximize the improvement in the objective function while ensuring that $\overline{x} \geq 0$. Observe from (2.1) and (2.2) that

$$\overline{x} = (\overline{x}_B, \overline{x}_P, \overline{x}_{N \setminus P}) = (x_B - \theta \overline{A}_{\cdot P} q_P, x_P + \theta q_P, x_{N \setminus P}). \tag{2.3}$$

Since $\overline{x}_{N\setminus P} = x_{N\setminus P} \geq 0$, only variables in P and B may violate their nonnegativity requirements. First, consider the variables in P. For $j \in P$,

$$\begin{aligned} x_j + \theta q_j \geq 0 &\Leftrightarrow \theta q_j \geq -x_j \\ &\Leftrightarrow \theta \leq -\frac{x_j}{q_j} \text{ for j: } q_j < 0 \\ &\Leftrightarrow \theta \leq \min_{j:q_j<0} -\frac{x_j}{q_j}. \end{aligned} \tag{2.4}$$

The same logic applies to the basic variables. For $j \in B$,

$$x_j + \theta q_j \geq 0 \Leftrightarrow \theta \leq \min_{j:q_j<0} -\frac{x_j}{q_j}. \tag{2.5}$$

In order for $\overline{x}$ to remain nonnegative, θ must satisfy (2.4) and (2.5) simultaneously. In other words,

$$\theta = \min_{j\in B\cup P:q_j<0} -\frac{x_j}{q_j}. \tag{2.6}$$

Recall that in the simplex method only a single nonbasic variable increases, so (2.4) is always true, and (2.6) simplifies to the usual ratio test on the basic variables. (2.6) guarantees that $\overline{x} = x + \theta q \geq 0$. Since $A\overline{x} = b$, $\overline{x}$ is a feasible solution for the linear programs (1.1) and (1.3). We now have defined a procedure to determine a search direction q and an associated step length θ.

The algorithm is incomplete since the rules governing changes in the basis and determination of promising variables and their rates of change remain undefined. Sections 2.3 and 2.4 examine those aspects in detail. Meanwhile, we extend the search procedure to allow for a second search direction $\hat{q}$. The algorithm will then use a linear combination of q and $\hat{q}$ to compute the next feasible iterate. We define $\hat{q}$ by viewing the constraints of the linear program (1.3) in the following equivalent form:

$$\begin{aligned} \overline{A}_{.P} x_P &\leq \overline{b} - \overline{A}_{.N\setminus P} x_{N\setminus P} \\ x_{N\setminus P},\, x_P &\geq 0. \end{aligned} \tag{2.7}$$

(2.7) considers the algorithm from the perspective of the space of the promising variables x_P. The basic variables x_B serve as slack variables. Observe that in this space, movement from x to $\overline{x}$ implies movement in search direction q until at least one of the hyperplanes defining the constraints is reached. In other words, once we determine a search direction q, suppose that we move from the current feasible solution x in the direction q as far as possible. The step length θ increases until a slack variable attains zero; (2.6) determines the size of the step length and identifies one or more such variables. A hyperplane corresponds to each slack driven to zero, and we shall utilize such a hyperplane to determine a second search direction. In order to identify a tight constraint, consider the ratio tests in (2.6). Let j^* index a variable achieving the minimum ratio, i.e.

$$j^* = \operatorname*{argmin}_{j\in B\cup P:q_j<0} -\frac{x_j}{q_j}.$$

Two cases arise. If $j^* \in P$, then $\overline{x}_{j^*} = 0$. Let r_P index the component of P containing j^* ; $P(r_P) = j^*$. Also, let $e^T_{r_P} \in R^{|P|}$ represent the $r_P{}^{th}$ unit row

vector. In this case a tight hyperplane corresponds to the nonnegativity constraint $e_{r_P}^T x_P \geq 0$ (or, equivalently, $x_{j^*} \geq 0$) found in (2.7). The second case occurs when $j^* \in B$. In this case suppose the $r_B{}^{th}$ component of B contains j^*. Then a tight hyperplane corresponds to the constraint $\overline{A}_{r_B,P} x_P \leq \overline{b}_{r_B} - \overline{A}_{\cdot N \setminus P} x_{N \setminus P}$. Summarizing the two cases, let $\overline{a}_P$ represent the normal to a tight hyperplane determined by the ratio tests in (2.6). Then,

$$\overline{a}_P^T = \begin{cases} e_{r_P}^T & \text{if } j^* \in P; \\ \overline{A}_{r_B,P} & \text{if } j^* \in B. \end{cases} \tag{2.8}$$

Given $\overline{a}_P$, project q_P, the promising components of the first search direction, onto the hyperplane $\overline{a}_P^T x_P = 0$ in order to generate a second direction $\hat{q}_P$. Recall that projection onto the null space of $\overline{a}_P^T$ is equivalent to projection onto the orthogonal complement of the row space of $\overline{a}_P^T$. Projecting onto the row space of $\overline{a}_P^T$ involves the projection matrix

$$\overline{a}_P(\overline{a}_P^T \overline{a}_P)^{-1} \overline{a}_P^T = \frac{\overline{a}_P \overline{a}_P^T}{\|\overline{a}_P\|_2^2}.$$

Projection onto the orthogonal complement of the row space of $\overline{a}_P^T$ involves the projection matrix

$$I - \frac{\overline{a}_P \overline{a}_P^T}{\|\overline{a}_P\|_2^2}.$$

Hence,

$$\begin{aligned} \hat{q}_P &= \left(I - \frac{\overline{a}_P \overline{a}_P^T}{\|\overline{a}_P\|_2^2}\right) q_P \\ &= q_P - \left(\frac{(\overline{a}_P^T q_P)}{\|\overline{a}_P\|_2^2}\right) \overline{a}_P. \end{aligned} \tag{2.9}$$

Notice that if $j^* \in P$, $\overline{a}_P$ is a unit vector, and the computation of $\hat{q}_P$ requires very little extra work:

$$\hat{q}_P = q_P - e_{r_P} q_{j^*}.$$

However, if $j^* \in B$, then

$$\hat{q}_P = q_P - \left(\frac{(e_{r_B}^T B^{-1} A_{\cdot P} q_P)}{\|e_{r_B}^T B^{-1} A_{\cdot P}\|_2^2}\right)(e_{r_B}^T B^{-1} A_{\cdot P})^T.$$

In this case one must solve the linear system $w^T B = e_{r_B}^T$ and then compute inner products between w and $A_{\cdot j}$ for each $j \in P$. These computations require significant extra work.

Given $\hat{q}_P$, we ensure that $\hat{q}$ lies in the null space of A by setting

$$\begin{aligned} \hat{q}_{N \setminus P} &= \quad 0 \quad \text{and} \\ \hat{q}_B &= -\overline{A}_{\cdot P} \hat{q}_P. \end{aligned} \tag{2.10}$$

Given a feasible solution x, we move to a new solution

$$\begin{aligned}\hat{x} &= (\hat{x}_B, \hat{x}_P, \hat{x}_{N\setminus P}) \\ &= (x_B - \theta \overline{A}_{\cdot P} q_P - \mu \overline{A}_{\cdot P} \hat{q}_P, x_P + \theta q_P + \mu \hat{q}_P, x_{N\setminus P}).\end{aligned} \tag{2.11}$$

Since $\hat{q}$ lies in the null space of A, $A\hat{x} = b$ for all values of θ and μ. We wish to choose θ and μ so that $\hat{x}$ satisfies the nonnegativity requirement while bringing about the largest possible change in the objective function. With a single search direction this occurs when θ is chosen as large as possible subject to the constraints specified by the ratio tests in (2.4) and (2.5); one can view this as a one-column linear program with solution given by (2.6). Finding the best combination of two search directions requires the solution of a two-column linear program:

$$\begin{aligned} \text{minimize} \quad & (\overline{c}_P^T q_P)\theta + (\overline{c}_P^T \hat{q}_P)\mu \\ \text{subject to} \quad & (-q_P)\theta + (-\hat{q}_P)\mu \le x_P \\ & (\overline{A}_{\cdot P} q_P)\theta + (\overline{A}_{\cdot P} \hat{q}_P)\mu \le x_B. \end{aligned} \tag{2.12}$$

Remember that x is a feasible solution, so θ and μ are the only variables involved in (2.12). The objective function in this linear program measures the change in the objective of the linear program (1.1) achieved by moving from x to $\hat{x}$. The first set of constraints correspond to the nonnegativity requirements on $\hat{x}_P$ implied by (2.11), while the second set corresponds to the analogous requirement for $\hat{x}_B$. Therefore, $\hat{x}$ is feasible for the linear programs (1.1) and (1.3).

We have just described a procedure that defines a two-dimensional search plane instead of the usual one-dimensional search line. The use of a search plane requires significant additional computation, but it offers an extra dimension that can help the algorithm avoid getting stuck in a long sequence of iterations with minimal progress. One can solve two-variable linear programs quickly. One could implement a specialized simplex method that exploits the fact that the constraint matrix of the linear program (2.12) consists of two columns plus an identity matrix. Therefore, during any iteration, a basis contains at most two columns other than unit vectors, which should reduce the work involved in basis factorizations and updates. A second approach consists of using a special linear-time algorithm for two-variable linear programming (see [30]). The latter method was adopted here.

2.3. The Choice of Promising Variables

Section 2.2 described how to change a feasible solution x to a new feasible solution $\overline{x}$ (or $\hat{x}$, if one uses two search directions). This section investigates the determination of P, the index set of promising variables. We wish to choose P so that a decrease in the objective function will always result from the change in x. In the literature the most frequently encountered selection rule consists of selecting each nonbasic variable that would individually decrease the objective function. In other words,

$$\begin{aligned} j \in P \text{ if } \quad & \overline{c}_j < 0,\ x_j \ge 0 \text{ or} \\ & \overline{c}_j > 0,\ x_j > 0. \end{aligned} \tag{2.13}$$

We then set $q_j = -\overline{c}_j$ for $j \in P$. (2.1) now precisely specifies the search direction q. Note that because x need not be a vertex, the objective function will improve by decreasing a positive nonbasic variable with positive reduced cost; such variables

do not exist in the simplex method. In Section 2.7 we will modify (2.13), but the change will not affect any of the ideas discussed in the remainder of this section.

Notice that the simplex method's one-to-one correspondence between bases and feasible solutions no longer exists. In fact, one can associate a feasible solution x with any basis, even an infeasible one. Since the values of the reduced costs depend on the choice of basis, so does the determination of P. Given x, a particular nonbasic variable could increase for one associated basis yet decrease for another. Nonetheless, the following lemma shows that q remains a descent direction regardless of the choice of basis.

Lemma 1. *Given a feasible solution x and any basis B for the linear program (1.1), suppose one defines q as follows:*

$$\begin{aligned} q_P &= -\bar{c}_P \\ q_{N\setminus P} &= 0 \\ q_B &= -\overline{A}_{\cdot P} q_P = \overline{A}_{\cdot P}\bar{c}_P. \end{aligned} \tag{2.14}$$

Then $c^T q < 0$.

Proof. We prove the lemma by partitioning c^T and q by B, P, and $N \setminus P$:

$$\begin{aligned} c^T q &= c_B^T q_B + c_P^T q_P + c_{N\setminus P}^T q_{N\setminus P} \\ &= c_B^T \overline{A}_{\cdot P}\bar{c}_P - c_P^T \bar{c}_P + 0 \\ &= c_B^T B^{-1} A_{\cdot P}\bar{c}_P - c_P^T \bar{c}_P \\ &= (\overbrace{\pi^T A_{\cdot P} - c_P^T}^{-\bar{c}_P^T})\bar{c}_P \\ &= -\|\bar{c}_P\|_2^2 < 0. \bullet \end{aligned}$$

Lemma 1 implies that a feasible descent direction always exists if the current solution is not optimal. Note that although the resulting second direction $\hat{q}$ need not guarantee descent, the optimal solution (θ^*, μ^*) to the two-column linear program (2.12) yields the descent direction $\theta^* q + \mu^* \hat{q}$ used to determine $\hat{x}$ in (2.11). This is true because one can set $\mu = 0$ in (2.12) and use only q as a search direction.

The loosening of the relationship between bases and feasible solutions discussed here raises a similar ambiguity with respect to optimality of a solution. One would hope that if the set P is empty during a particular iteration, then the current feasible solution is in fact optimal. Lemma 2 reveals this to be true regardless of the associated basis.

Lemma 2. *Given a feasible solution x and a basis B for the linear program (1.1), if P is empty, then x is optimal.*

Proof. We prove the lemma by utilizing the complementary slackness conditions, which are necessary and sufficient for optimality. First, note that for $j \in B$, $\bar{c}_j = 0$ and $x_j \geq 0$. Furthermore, since P is empty, its definition (2.13) implies that for $j \in N$, $x_j = 0$ if $\bar{c}_j > 0$, and $\bar{c}_j = 0$ if $x_j > 0$. We conclude that the primal feasible solution x and the dual feasible solution $(\pi, \bar{c}\,)$ satisfy the complementary slackness conditions. •

Lemma 2 provides a simple termination criterion for the algorithm. Notice that, unlike the simplex method, the resulting optimal solution need not be a vertex.

When solving practical linear programs, the modeller frequently wishes to perform sensitivity analysis. In order to do so, the optimal basis must correspond directly with the optimal solution. Unlike the simplex method, the algorithm described here does not guarantee this correspondence. Fortunately, one can design a simple procedure to move from an optimal solution to an optimal vertex. For details refer to the appendix.

2.4. Pivoting Strategies

After defining the set of promising variables and moving to a new feasible solution, all that remains undefined in an iteration is a pivoting procedure to change the basis. For expository purposes assume the algorithm uses a single search direction q; the procedure is very similar for two search directions. Many different pivoting strategies are available. For the computational tests in Section 2.8, the pivot rule depends on the value of the step length θ defined by (2.6).

Case 1: Suppose that $\theta > 0$. Given the precise definition of q in (2.14), we specify the ratio test in (2.6):

$$\theta = \min_{j:q_j<0} -\frac{x_j}{q_j} = \min\left\{ \min_{j\in P:\bar{c}_j>0} \frac{x_j}{\bar{c}_j}, \min_{i:\overline{A}_{i,P}\bar{c}_P<0} -\frac{x_{j_i}}{\overline{A}_{i,P}\bar{c}_P} \right\}. \tag{2.15}$$

Recall that $\bar{x}$ is the current feasible solution resulting from the iterative step (2.2). Let s index the entering basic variable; choose the largest promising variable to enter the basis:

$$s = \operatorname*{argmax}_{j\in P} \bar{x}_j. \tag{2.16}$$

Other criteria were also tested, but (2.16) emerged from the experiments as the best one.

Selection of the outgoing variable depends on the sign of the reduced cost $\bar{c}_s$. If $\bar{c}_s < 0$, we wish to increase the incoming variable. However, if the ratio test in (2.15) results in a basic variable reaching zero, we also wish to remove that variable from the basis. The following procedure always achieves at least one of these goals. As in Section 2.2, let r_B be a component of B containing a basic variable driven to zero by the basic variable ratio test in (2.15); j_{r_B} indexes the corresponding variable. Define $\overline{A}_{\cdot s} = B^{-1}A_{\cdot s}$ as the representation of the incoming column with respect to the current basis. If $\overline{A}_{r_B,s} > 0$, then select j_{r_B} as the outgoing basic variable and change the basis. Otherwise, select the outgoing variable by the usual simplex method ratio test:

$$r = \operatorname*{argmin}_{i:\overline{A}_{i,s}>0} \frac{x_{j_i}}{\overline{A}_{i,s}}. \tag{2.17}$$

Choose the r^{th} basic variable j_r to leave the basis. Associated with r is $\bar{\theta}$, the increase in the entering variable:

$$\bar{\theta} = \min_{i:\overline{A}_{i,s}>0} \frac{x_{j_i}}{\overline{A}_{i,s}}. \tag{2.18}$$

Note that if $\overline{A}_{i,s} \leq 0$, the algorithm terminates with an unbounded solution. If not, update the current feasible solution:

$$\bar{x}_s \leftarrow \bar{x}_s + \bar{\theta}$$

$$\bar{x}_B \leftarrow \bar{x}_B - \overline{A}_{\cdot s}\bar{\theta}.$$

Note that the updated solution remains feasible. Now, replace the r^{th} column of the basis with $A_{\cdot s}$. At this point a few observations are in order. First of all, note that if r_B exists and $\overline{A}_{r_B,s} > 0$, then r_B is an argmin of the standard ratio test (2.17), and the corresponding increase $\overline{\theta}$ is zero. Note also that if no basic variable hits zero in the ratio test (2.15), then no such index r_B exists; proceed immediately with the ratio test (2.17). This completes the pivot rule when $\overline{c}_s < 0$.

If $\overline{c}_s > 0$, we wish to decrease the incoming variable as much as possible without violating its nonnegativity requirement. One must also ensure that the basic variables remain nonnegative. In this case, if r_B exists and $\overline{A}_{r_B,s} < 0$, then the r_B^{th} basic variable leaves the basis; do not perform a ratio test. Otherwise, perform the following ratio test to preserve nonnegativity of the basic variables:

$$r = \underset{i:\overline{A}_{i,s}<0}{\operatorname{argmin}} -\frac{x_{j_i}}{\overline{A}_{i,s}}. \tag{2.19}$$

The difference between (2.17) and (2.19) occurs because the entering variable decreases when $\overline{c}_s < 0$. In addition, define

$$\overline{\theta} = \min_{i:\overline{A}_{i,s}<0} -\frac{x_{j_i}}{\overline{A}_{i,s}}. \tag{2.20}$$

$\overline{\theta}$ is the largest possible decrease in the entering variable that preserves nonnegativity of the basic variables. Remember that one must also guarantee that the decrease in the entering variable does not violate its nonnegativity constraint. Observe that this implies that the algorithm never terminates with an unbounded solution when $\overline{c}_s < 0$. If $\overline{\theta} \geq \overline{x}_s$, the entering variable reaches zero without driving any of the basic variables below zero. In this case avoid the pivot and merely update the present solution:

$$\overline{x}_s \leftarrow 0$$
$$\overline{x}_B \leftarrow \overline{x}_B + \overline{A}_{\cdot s}\overline{x}_s.$$

If $\overline{\theta} < \overline{x}_s$, then a pivot is necessary. Update $\overline{x}$ as follows:

$$\overline{x}_s \leftarrow \overline{x}_s - \overline{\theta}$$
$$\overline{x}_B \leftarrow \overline{x}_B + \overline{A}_{\cdot s}\overline{\theta}.$$

Replace $A_{\cdot j_r}$ with $A_{\cdot s}$ as the r^{th} basic column. This completes the pivoting procedure when $\overline{c}_s > 0$.

Case 2: Suppose that θ, the step length in (2.15), is zero. Observe that promising variables with negative reduced costs cannot directly bound θ. Since $x_j > 0$ if $j \in P$ and $\overline{c}_j > 0$, one sees that among the promising variables the minimum ratio in (2.15) must be positive. It follows that $\theta = 0$ only if at least one basic variable j_r is zero and the corresponding component in the search direction $q_{j_r} = \overline{A}_{r,P}\overline{c}_P < 0$. Experimentation with a set of 12 practical problems suggested the benefit of removing such variables from the basis. Accordingly, a procedure to ensure removal was adopted. Define

$$S = \{j \in P : \overline{c}_j < 0, \overline{A}_{r,j} > 0 \quad \text{or}$$
$$\overline{c}_j > 0, \overline{A}_{r,j} < 0\}.$$

Since $\overline{A}_{r,P}\bar{c}_P < 0$, we know that $\exists\ j \in P : \bar{c}_j < 0$ and $\overline{A}_{r,j} > 0$, or $\bar{c}_j > 0$ and $\overline{A}_{r,j} < 0$. Hence, $|S| \geq 1$. In other words, there exists at least one promising variable whose entrance into the basis will remove the r^{th} basic variable. S indexes all such variables. We now select the largest nonbasic variable in S to enter the basis:

$$s = \operatorname*{argmax}_{j \in S} \overline{x}_j.$$

One can improve the stability of the algorithm by modifying the rule to ensure a sufficiently large pivot element. Murtagh and Saunders [32] recommend the following modification. Define

$$\alpha = \max_{j \in S} |\overline{A}_{r,j}|,$$

and choose

$$s = \operatorname*{argmax}_{j \in S} \{\overline{x}_j : |\overline{A}_{r,j}| \geq 0.1\alpha\}.$$

Observe that if $\bar{c}_s < 0$, then $\overline{A}_{r,s} > 0$ since $s \in S$; hence we see from the standard ratio test (2.17) that the r^{th} basic variable is eligible to leave the basis. Similarly, the ratio test (2.19) assures eligibility if $\bar{c}_s > 0$. In each case $\bar{\theta}$, the change in the incoming variable, is zero, so no change in the current feasible solution occurs. The pivoting procedure when $\theta = 0$ is now complete.

We have now accounted for all possible values of θ. The pivot rule is complete. We can now update the basis and proceed with the next iteration.

2.5. Effects of Degeneracy

The previous three sections provided a general description of a type of feasible direction method for linear programming. Some flexibility exists with respect to selection of promising variables, pivoting procedure, and other details, but the fundamental approach remains unchanged.

Very little computational testing of this type of algorithm exists in the literature. Nonetheless, several publications remain noteworthy. In 1979 Cooper and Kennington [5] discussed linear programming algorithms:

> We find it curious that the literature contains so few papers concerning other algorithms for such an important class of problems. We assume either (i) other ideas have been investigated, abandoned, and never reported, or (ii) the simplex method has proved so effective that other investigators felt no motivation to work in this area.

They went on to propose algorithms within the class described here. No computational testing was performed. Sherali, Soyster, and Baines [45] tested a similar algorithm on a set of random problems. They remarked:

> Computationally, this method turned out to be substantially inferior to the simplex method ... One may expect in this instance that after some rapid initial improvements, the reduced gradient procedure goes through many more insignificant iterations. This was not the case ... What appears to happen is that instead of jumping along the simplex path, and hence rendering itself advantageous, the procedure zigzags across the simplex path, resulting in several more iterations.

Eiselt and Sandblom [8] also report discouraging results for a similar approach:

> The intended "shooting through polytopes" in our study resulted in many problems, most prominently numerical instability and convergence problems. On that basis, the method was referred to as "crawling and stalling" and work on it was discontinued.

Eiselt and Sandblom altered their approach to allow for the notion of external pivoting. With this modification they reported encouraging computational results on a set of random problems. Meanwhile, Kallio and Orchard-Hays [22] tested a reduced-gradient method on a set of non-trivial practical problems. They restricted the set of promising variables to at most seven, regardless of problem size. They also utilized a multiple pricing procedure in an attempt to reduce the average work per iteration. Nonetheless, they too found their approach required more iterations than the simplex method on most of their test problems.

The initial tactic adopted here was to utilize the two-dimensional search method described in Section 2.2. It was hoped that this method would alleviate the difficulties described by these authors. However, the flexibility provided by an additional direction proved insufficient to make the approach competitive with the simplex method. Iteration counts ranged from 1.3 to 2.1 times those of the simplex method on the 12 practical problems tested. CPU times were not even considered since each iteration requires significant extra work compared to a simplex iteration.

The reader may find these results surprising; intuitively one might anticipate that moving through the feasible region instead of around it would provide a substantial advantage over the simplex method. However, detailed examination of these algorithms reveals an explanation for the poor performance on practical problems.

Sparsity and degeneracy are two characteristics of practical problems that are absent from most randomly generated problems. Although one can generate sparse, random problems, they still lack the sparsity patterns characteristic of real problems. Sparsity and degeneracy inhibit the performance of the algorithms of Sections 2.2–2.4. In particular, values of zero in the ratio test that determines the step length occur much more frequently than in the simplex method. To see why, recall the ratio tests involved to determine the step lengths. If we use a single search direction, define P by (2.13), and set $q_j = -\overline{c}_j$ for $j \in P$, then the step length θ is bounded above (see (2.6) and (2.15)) by the following minimum ratio:

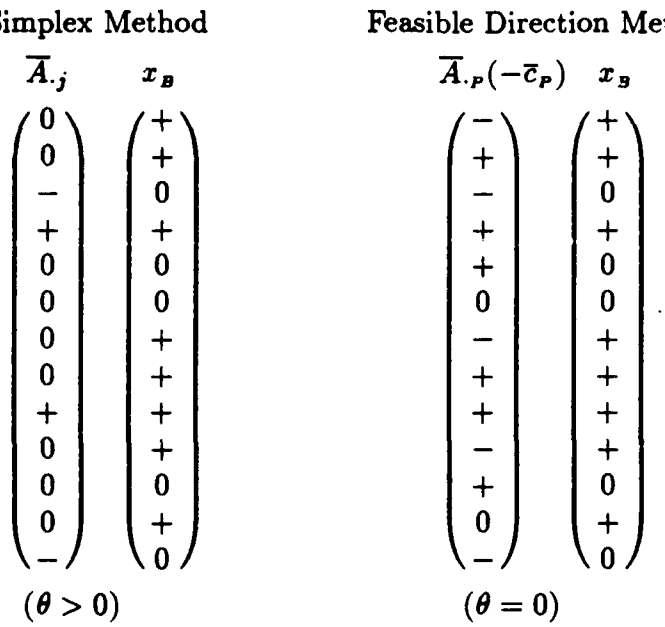
$$\theta \leq \min_{i:\overline{A}_{i,P}(-\overline{c}_P)>0} \frac{x_{j_i}}{\overline{A}_{i,P}(-\overline{c}_P)}. \tag{2.21}$$

On the other hand, a slightly different ratio test bounds the step length in the simplex method:

$$\theta \leq \min_{i:\overline{A}_{i,s}>0} \frac{x_{j_i}}{\overline{A}_{i,s}}. \tag{2.22}$$

Note that the simplex method uses a single column $\overline{A}_{\cdot s}$ in the ratio test, while the feasible direction method uses a linear combination of many such individual columns. Why is this difference significant? On problems tested here the canonical columns $\overline{A}_{\cdot j}$ typically remained sparse, albeit not to the extent of the corresponding original columns $A_{\cdot j}$. Taking a linear combination of many such sparse columns as in the ratio test (2.21) increases the density of the composite column $\overline{A}_{\cdot P}(-\overline{c}_P)$. This occurs regardless of the choice of search direction if $|P|$ is substantial. The result, as illustrated by Figure 2-1, is that the dense composite column contains

Figure 2-1. Degeneracy and Linear Programming Algorithms



many positive components, each of which is eligible in the ratio test. Contrast this with the simplex method, where usually only a few positive components exist.

Given the presence of degeneracy in the basic variables, one sees that the minimum ratio of (2.21) equals zero with greater likelihood than that of (2.22). This also explains why the simplex method can frequently solve degenerate linear programs without performing too many degenerate pivots; the degenerate basic variables frequently correspond to non-positive components of the column in the ratio test. The same cannot be said for the algorithms in Sections 2.2–2.4.

This kind of difficulty need not arise only in the presence of degeneracy. As long as the nonbasic columns $\overline{A}_{\cdot j}$ remain sparse, feasible direction methods will frequently be restricted by the step length associated with the worst promising column. To see this, consider the following small numerical example:

$$\overline{A}_{\cdot 1} = \begin{pmatrix} 1 \\ 2 \\ -2 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \overline{A}_{\cdot 2} = \begin{pmatrix} 0 \\ -1 \\ 1 \\ -1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \overline{A}_{\cdot 3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 8 \end{pmatrix} \qquad x_B = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 4 \\ 2 \\ \epsilon \\ 8 \end{pmatrix}$$

$$\overline{c}_{\{1,2,3\}} = (-1, -1, -\frac{1}{2}). \tag{2.23}$$

In this example $P = \{1, 2, 3\}$, and $0 \leq \epsilon < 1$. In the simplex method the step lengths associated with entering columns 1,2, and 3 into the basis are 1,1, and ϵ, respectively. The corresponding improvements in objective function are 1,1, and $\frac{\epsilon}{2}$. Since ϵ may be arbitrarily small, we see that columns 1 and 2 are good choices,

while column 3 is a poor one. Fortunately, the standard simplex method would choose 1 or 2 to enter the basis in this situation. Given the choice of P, we see that

$$\overline{A}_{\cdot P}(-\overline{c}_P) = \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 0 \\ \frac{1}{2} \\ 4 \end{pmatrix}.$$

From the ratio test (2.21) it follows that the corresponding step length cannot exceed 2ϵ. Similarly, since $\overline{c}_P^T q_P = -\overline{c}_P^T \overline{c}_P = -\frac{9}{4}$, the resulting improvement in the objective function is at most $\frac{9}{2}\epsilon$. Thus, for $\epsilon < \frac{2}{9}$, the feasible direction algorithm yields a smaller improvement than the simplex method. Note, however, that if $P = \{1, 2\}$, then

$$\overline{A}_{\cdot P}(-\overline{c}_P) = \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

The resulting step length is now 1 and the improvement in the objective function is 2, so the feasible direction approach outperforms the simplex method. The important observation here is that $\overline{A}_{\cdot 3}$ is complementary to $\overline{A}_{\cdot 1}$ and $\overline{A}_{\cdot 2}$. Therefore, $\overline{A}_{\cdot 3}$ limits the feasible direction step length regardless of the benefit of other promising columns. Because of sparsity, this situation occurs frequently in practical problems, and it creates a significant obstacle for any of the algorithms previously described herein. Of course, the simplex method, which is a feasible direction method that selects only one promising variable, may also choose poorly. However, the point here is that it takes only one bad column to inhibit the step length; an algorithm selects such a column more frequently when it chooses many promising variables instead of one.

How can we overcome these difficulties? Observe that in the numerical example the feasible direction method progresses nicely when we exclude column 3 from the set of promising variables. This suggests the benefit of screening out certain bad columns as unpromising, even though they satisfy the previous promising criterion (2.13). In the next section we shall develop some computationally inexpensive techniques to do so.

2.6. Dynamic Pricing

Section 2.5 demonstrated how a single column with a small step length can inhibit progress of the algorithms of Sections 2.2–2.4. We now develop methods to screen out such columns in a computationally inexpensive way.

Let us begin by considering columns with zero step lengths. Define

$$\theta_j = \min_{i:\overline{A}_{i,j}>0} \frac{x_{j_i}}{\overline{A}_{i,j}} \qquad j \in N. \tag{2.24}$$

θ_j is the step length that would result from increasing the j^{th} variable. For expository purposes we shall consider the case of promising variables with negative reduced costs; similar logic applies for those with positive reduced costs. For a variable with a zero step length, $\theta_j = 0$, so from (2.24) one sees that at least one basic variable equals zero and corresponds to a positive component of $\overline{A}_{\cdot j}$ in the ratio test. How can one detect such columns in advance? For each $j \in P$, one could explicitly determine θ_j and exclude any columns for which $\theta_j = 0$. Unfortunately, this would require the representation $\overline{A}_{\cdot P}$ of the columns $A_{\cdot P}$ in terms of the current basis. In other words, one must solve $|P|$ systems of linear equations of the form $By = A_{\cdot j}$, a prohibitively expensive task. A much cheaper approach consists of defining a second objective function that measures the degeneracy of the basic variables in a way that yields valuable information on how to exclude bad columns. Since the basic variables change during iterations when the objective improves, this second objective function changes during the course of the algorithm. In particular, at the start of a given iteration, define $d \in R^n$ so that $d_j = 0$ for $j \in N$, and

$$d_{j_i} = \begin{cases} 1 & \text{if } x_{j_i} = 0, \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

for $i = 1, \ldots, m$. Thus, d identifies the degenerate basic variables. For $j \in F$ compute the quantity $\overline{d}_j = d_B^T \overline{A}_{\cdot j}$. The following lemma provides a simple column screening criterion.

Lemma 3. *If $\overline{d}_j > 0$, then $\theta_j = 0$.*

Proof. A closer look at $\overline{d}_j$ proves the lemma. Observe from the definition of d_B in (2.25) that

$$\overline{d}_j = \sum_{i: x_{j_i} = 0} \overline{A}_{i,j}.$$

In other words, $\overline{d}_j$ consists of the sum of components of $\overline{A}_{\cdot j}$ that correspond to a degenerate basic variable in the ratio test in (2.24). Therefore, if $\overline{d}_j > 0$,

$$\exists\, i^* : \overline{A}_{i^*,j} > 0,\ x_{j_{i^*}} = 0.$$

Note that i^* is eligible for the ratio test (2.24), so

$$0 \le \theta_j \le \frac{x_{j_{i^*}}}{\overline{A}_{i^*,j}} = 0,$$

which establishes the desired result. •

Thus, if $\overline{d}_j > 0$, one knows in advance that x_j results in a zero step length if it becomes basic. Furthermore, we shall see how to compute $\overline{d}_j$ efficiently. We now have a method to exclude columns from P. Note that $\overline{d}_j < 0$ does not necessarily imply that $\theta_j > 0$; consider

$$\overline{A}_{\cdot j} = \begin{pmatrix} 1 \\ -2 \\ -2 \\ 0 \\ 0 \end{pmatrix}, \quad x_B = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 4 \\ 0 \end{pmatrix}.$$

We now show that the computation of $\overline{d}_j$ requires essentially the same amount of work as computing the reduced costs $\overline{c}_j$. One does not compute $\overline{d}_j$ from the expression $d_B^T \overline{A}_{\cdot j}$; this would involve solving for each $\overline{A}_{\cdot j}$, $j \in P$. Instead, observe that

$$\overline{d}_j = d_B^T \overline{A}_{\cdot j} = (d_B^T B^{-1}) A_{\cdot j}.$$

Let $\sigma^T = d_B^T B^{-1}$. Generate σ by solving the linear system

$$\sigma^T B = d_B^T. \tag{2.26}$$

Solving for σ is analogous to solving for π, the dual variables. Now, compute $\overline{d}_j = \sigma^T A_{\cdot j}$. Computing $\overline{d}_j$ for all $j \in P$ requires only one additional solve, as opposed to the $|P|$ additional solves needed to examine individual components of each $\overline{A}_{\cdot j}$. More generally, one can generate any linear combination of the rows of the matrix $\overline{A}_{\cdot P}$ with one additional solve, but examination of individual components of each column of that matrix requires $|P|$ extra solves. The key, of course, consists of choosing a linear combination that yields valuable information, as in (2.25).

Let us now extend this notion to deal with small values of θ_j. In (2.25) an indicator function $I_{\{x_{j_i}=0\}}$ determined the value of d_{j_i}. In general one can set

$$d_{j_i} = f(x_{j_i}) \quad i = 1, \ldots, m \tag{2.27}$$

for any specified function f. Lemma 4 proposes some beneficial choices of f.

Lemma 4. *Given $\tau > 0$, suppose that $d_{j_i} = f(x_{j_i})$ where*

$$\begin{aligned} &f(x_{j_i}) > 0 \text{ if } x_{j_i} < \tau, \\ &f(x_{j_i}) = 0 \text{ if } x_{j_i} \geq \tau. \end{aligned} \tag{2.28}$$

Then

$$\overline{d}_j > 0 \Rightarrow \theta_j < \frac{\tau}{\min\limits_{i:\overline{A}_{i,j}>0} \overline{A}_{i,j}}.$$

Proof. Lemma 3 is a special case of Lemma 4, so we utilize a similar strategy to prove the result. Given the definition of f,

$$\overline{d}_j = \sum_{i:x_{j_i}<\tau} \overline{A}_{i,\cdot} f(x_{j_i}) > 0 \Rightarrow \exists\, i^* : \overline{A}_{i^*,j} > 0 \text{ and } x_{j_{i^*}} < \tau.$$

The basic variable indexed by i^* is eligible for the ratio test in (2.24), so

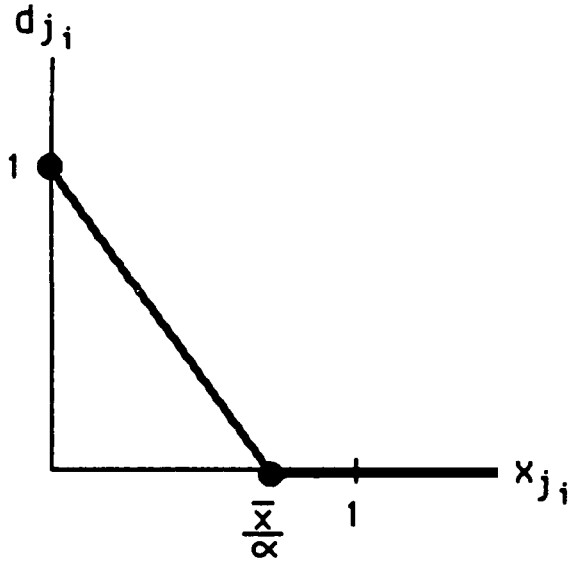
$$0 \leq \theta_j \leq \frac{x_{j_{i^*}}}{\overline{A}_{i^*,j}} \leq \frac{x_{j_{i^*}}}{\min\limits_{i:\overline{A}_{i,j}>0} \overline{A}_{i,j}} < \frac{\tau}{\min\limits_{i:\overline{A}_{i,j}>0} \overline{A}_{i,j}}. \bullet$$

Lemma 4 provides a way to screen out columns with small step lengths. Again, the procedure requires only one additional solve. The particular choices of f and τ are important. One can specify a piecewise linear function to satisfy the condition

(2.28) of Lemma 4. For example, define $\overline{x} = \frac{1}{m}\sum_{i=1}^{m} x_{j_i}$ as the average of the basic variables, and let α represent a positive scalar. Set

$$d_{j_i} = f(x_{j_i}) = \left[1 - \frac{\alpha x_{j_i}}{\overline{x}}\right]^+. \tag{2.29}$$

Consider the plot of d_{j_i} against x_{j_i}.



As in (2.25), we again set $d_{j_i} = 1$ if $x_{j_i} = 0$, but we now utilize the values of $\overline{x}$ and α to account for relatively small basic variables.

For an illustration, set $\alpha = 1$ and reconsider example (2.23) with $\epsilon = .1$. One sees that $\overline{d}_3 > 0$, resulting in the exclusion of column 3 from P. The feasible direction method then achieves a greater improvement in the objective function than the simplex method. One can also construct examples of good columns failing the screening test and bad columns passing it. Nonetheless, this choice of f has proven useful in practice.

We now have developed an inexpensive way to evaluate further the promise of potential entering variables. In the next section we shall use these ideas to propose a modified feasible direction algorithm. Later we will extend these ideas and apply them to the simplex method.

2.7. A Modified Feasible Direction Method

The results of Sections 2.5 and 2.6 suggest a slightly different feasible direction method. In particular, in light of Lemmas 3 and 4, define $P^* \subseteq P$ as follows:

$$\begin{aligned} j \in P^* \text{ if } \quad & \overline{c}_j < 0,\ x_j \geq 0,\ \overline{d}_j \leq 0 \text{ or} \\ & \overline{c}_j > 0,\ x_j > 0,\ \overline{d}_j \geq 0. \end{aligned} \tag{2.30}$$

In other words, a variable remains promising only if it passes one of the screening criteria of Section 2.6. Notice that for positive variables in P, negative values of

Figure 2-2. A Modified Feasible Direction Method

Given: a basis B and a feasible solution x for the linear program (1.1).

1. Determine the sets P and P^* by (2.13) and (2.30).
2. If $P = \emptyset$, go to 10.
3. If $P^* \neq \emptyset$, set $P = P^*$.
4. Determine the search direction q: $q_P = -\overline{c}_P$, $q_{N \setminus P} = 0$, $q_B = \overline{A}_{\cdot P}\overline{c}_P$.
5. Determine the step length θ :

$$\theta = \min_{j \in B \cup P : q_j < 0} -\frac{x_j}{q_j}.$$

6. Move to a new feasible solution $\overline{x} = x + \theta q$.
7. Determine the incoming and outgoing basic variables by the rules of Section 2.4. If the procedure reveals an unbounded solution, go to 11.
8. Update the basis.
9. Go to 1.
10. The current solution is optimal.
11. Terminate the algorithm.

$\overline{d}_j$ identify excluded columns. Recall that when a variable decreases, the negative components of $\overline{A}_{\cdot j}$ become eligible for the ratio test (2.20). This change in sign explains the different interpretation of $\overline{d}_j$ when a variable decreases.

We now substitute P^* for P in the previously described algorithms. The results remain unchanged except for the termination criterion. Emptiness of P^* need not imply optimality of the current solution. Emptiness of P remains as the stopping rule. Figure 2-2 summarizes the modified algorithm using a single search direction. The changes do not affect the theory of the reduced-gradient method, so the algorithm attains an optimal solution if one exists.

Computational tests revealed the effectiveness of the column screening techniques of Section 2.6. Different choices of the second objective function d were examined. Each one substantially reduced the iteration counts. The best approach discovered so far used the piecewise linear function in (2.29) with $\alpha = 10$. The use of a second search direction $\hat{q}$ provided only a marginal reduction in iterations that failed to compensate for the extra work involved. Screening out "bad" columns emerged as the most important enhancement to the performance of this type of algorithm.

2.8. Computational Results

We now examine some computational results. The test set consists of 12 small to moderately sized practical problems available from the Systems Optimization Laboratory at Stanford University. MINOS 5.1, a linear and nonlinear optimization code developed by Murtagh and Saunders (see [33] and [34]), figured prominently in the testing. The algorithm of Figure 2-2 was implemented by modifying the appropriate subroutines of MINOS. MINOS also provided the simplex method used in the comparisons. Identical subroutines performed many of the common aspects of each algorithm, including input of the problems, basis factorization, and solution of linear systems of equations. Thus, one can attribute distinctions in performance to different characteristics of the algorithms, instead of inconsistencies in

Figure 2-3. Results for Feasible Direction Method on Unscaled Problems

	ITERATIONS			CPU		
Problem	Feas. Dir.	Simplex	Feas./Simp	Feas. Dir.	Simplex	Feas./Simp
AFIRO	6	6	1.00	2.55	2.50	1.02
SHARE2B	38	48	0.79	18.17	13.55	1.34
BEACONFD	57	54	1.06	40.66	27.03	1.50
CAPRI	104	114	0.91	104.87	72.27	1.45
BRANDY	173	118	1.47	146.53	71.42	2.05
ADLITTLE	109	126	0.87	21.19	13.43	1.58
SHARE1B	139	169	0.82	75.34	40.50	1.86
ISRAEL	255	256	1.00	132.99	60.59	2.19
BANDM	225	273	0.82	226.55	134.18	1.69
STAIR	207	274	0.76	440.90	251.21	1.76
ETAMACRO	261	335	0.78	268.83	180.22	1.49
E226	410	493	0.83	255.39	137.79	1.85
		Geom. mean:	0.91		Geom. mean:	1.62

the implementations.

Figure 2-3 summarizes the experiments. We compare both iterations and CPU time. Iterations pertain to Phase II only; Phase I of the simplex method generated the same feasible vertex for the feasible direction and simplex methods. Time comparisons measure solution of the whole problem. In each case we compute the ratio of computational effort required by the feasible direction method to that of the simplex method. Ratios less than 1.0 identify superior performance by the feasible direction method. At the bottom of the table we compute the geometric mean of the 12 ratios. Assuming all problems are equally important, this measures relative performance accumulated over all test problems. With respect to iterations, the algorithm achieved moderate success. Most problems required less iterations than the simplex method; only BRANDY needed substantially more. However, each iteration involves significant additional work, and the reduction in iterations failed to compensate for this. CPU times exceeded those of the simplex method for every problem.

What conclusions can one draw from these results? The pricing techniques of Section 2.6 improve the algorithm noticeably. Iterations are usually fewer than those of the simplex method. This marks a significant improvement over the results of previous testing of this type of algorithm. Nonetheless, the approach still fails to compete with the simplex method on the basis of computation time, the most important indicator. The flexibility of the search direction and pivot rules suggests that more successful variants may exist. However, the research reported here reveals certain inherent difficulties with the approach. Column screening is beneficial, but the procedures defined are not flawless; columns with small step lengths can pass the screening test and inhibit the step length. The likelihood of such columns slipping through and being included amongst the promising variables is greater when $|P|$ is large than when $|P| = 1$, as in the simplex method. This fact remains true regardless of the particular choice of search direction or pivoting strategy. In order to succeed, any member of this class of algorithms must contain features designed to evade this obstacle.

CHAPTER 3: MULTIPLE-OBJECTIVE PIVOT RULES IN THE SIMPLEX METHOD

3.1. Preliminaries

The results of Chapter 2 suggest the potential of applying a two-objective approach to the simplex method. The simplex method tends to perform poorly on highly degenerate linear programs, so the ability to avoid degenerate pivots may be quite useful. Section 3.2 utilizes the results of Chapter 2 to formulate pivot rules for the simplex method. Section 3.3 then extends these ideas. Instead of trying to exclude certain variables, we investigate the use of a second objective function to make good selections. Two more pivot rules arise. Further examination reveals that these procedures attempt to estimate inexpensively the step length associated with a potential entering variable.

Section 3.4 examines a parametric variant of the simplex method which has performed well on highly degenerate test problems. The variant resembles the other pivot rules of this chapter because it also utilizes a second objective function, albeit one that remains unchanged throughout the algorithm. The similarities motivate a new parametric algorithm that incorporates dynamic pricing. Section 3.5 then considers the extra work required by the two-objective approach. The chapter concludes with the development of techniques to reduce the additional computation.

3.2. Column Screening in the Simplex Method

Although motivated by feasible direction methods, Lemmas 3 and 4 apply directly to the simplex method. Instead of choosing promising variables by (2.30), we formulate a two-priority procedure to select the incoming variable. The following criterion helps avoid degenerate pivots:

$$\begin{aligned} d_N &= 0, \\ d_{j_i} &= I_{\{x_{j_i}=0\}} \qquad i = 1, \ldots, m; \\ \text{First Priority: } s &= \operatorname*{argmin}_{\bar{c}_j<0, \bar{d}_j \le 0} \bar{c}_j, \\ \text{Second Priority: } s &= \operatorname{argmin} \bar{c}_j. \end{aligned} \tag{3.1}$$

In other words, select a variable that passes the screening test defined by Lemma 3. If none exist, use the standard selection rule. The incoming variable always has a negative reduced cost, and termination occurs only when all nonbasic variables have nonnegative reduced costs. Therefore, assuming it uses a suitable technique to resolve degeneracy, the simplex method will obtain an optimal solution in a finite number of iterations.

Dantzig, Wolfe and Bland (see [6], [51], and Shamir [44]) proposed pivot rules to handle degeneracy. The intent of these criteria was to establish finite behavior of the simplex method. The first two rules use a very specific procedure to define the outgoing variable, while the choice of incoming variable is arbitrary amongst those with negative reduced cost. Bland's rule explicitly determines both the incoming and the outgoing variables. The pivot rule (3.1) provides no guarantee of convergence unless accompanied by a suitable degeneracy resolution technique. To see this, refer to Hoffmann's cycling example in [6]. Nonetheless, (3.1) differs from the

other rules because it deals directly with degeneracy during the selection process. It tries to avoid problems with degeneracy instead of resolving them after their occurrence.

Lemma 4 and the piecewise linear function of (2.29) motivate a pivot rule identical to (3.1). The only difference is in the choice of d:

$$\begin{aligned} d_N &= 0, \\ d_{j_i} &= \left[1 - \frac{\alpha x_{j_i}}{\overline{x}}\right]^+ \qquad i = 1, \ldots, m; \\ \text{First Priority: } s &= \operatorname*{argmin}_{\overline{c}_j < 0, \overline{d}_j \leq 0} \overline{c}_j, \\ \text{Second Priority: } s &= \operatorname*{argmin} \overline{c}_j. \end{aligned} \tag{3.2}$$

This rule attempts to avoid small pivot steps. Once again, the simplex method obtains an optimal solution in a finite number of iterations.

3.3. Estimating the Step Length

The pivot rules of the preceding section use a second reduced cost $\overline{d}_j$ to avoid poor choices of incoming variable. We now attempt to use $\overline{d}_j$ to select variables with large step lengths. In fact, for a certain choice of d, a direct connection between $\overline{d}_j$ and the step length θ_j emerges.

To begin, define d as in the previous pivot rule (3.2). Consider the following three-tiered pivot rule:

$$\begin{aligned} \text{First Priority: } s &= \operatorname*{argmin}_{\overline{c}_j < 0, \overline{d}_j = 0} \overline{c}_j, \\ \text{Second Priority: } s &= \operatorname*{argmax}_{\overline{c}_j < 0, \overline{d}_j < 0} \overline{c}_j \overline{d}_j, \\ \text{Third Priority: } s &= \operatorname*{argmin}_{\overline{c}_j < 0, \overline{d}_j > 0} \frac{\overline{c}_j}{\overline{d}_j}. \end{aligned} \tag{3.3}$$

What motivates such a rule? The value of $\overline{d}_j$ provides information on the components of $\overline{A}_{\cdot j}$ that correspond in the simplex method ratio test (2.24) to basic variables smaller than $\overline{x}/\alpha$. Lemma 4 establishes the undesirability of nonbasic variables with positive values of $\overline{d}_j$. Larger positive values are even worse since they imply the presence of either more positive components of $\overline{A}_{\cdot j}$ or a few large positive components. Each of these occurences suggests a small step length. If $\overline{d}_j > 0$, use the ratio $\overline{c}_j/\overline{d}_j$ to incorporate information from both reduced costs. This approach balances the good aspects of more negative values of $\overline{c}_j$ with the unfavorable aspects of large positive values of $\overline{d}_j$. Similarly, negative values of $\overline{d}_j$ suggest the prevalence of negative values in the components of $\overline{A}_{\cdot j}$ involved in the ratio test. Only positive components of $\overline{A}_{\cdot j}$ can bound θ_j, so nonbasic variables with more negative values of $\overline{d}_j$ are less likely to have small step lengths. The quantity $\overline{c}_j \overline{d}_j$ estimates the improvement in the objective function if x_j enters the basis. Why should variables with $\overline{d}_j = 0$ receive top priority? The particular choice of d

in (3.2) motivates this distinction. Note that one determines d_B independently of $\overline{A}_{\cdot j}$. Since $\overline{d}_j = d_B^T \overline{A}_{\cdot j}$, and $d_{j_i} = 0$ if $x_{j_i} \geq \overline{x}/\alpha$, one anticipates that if $\overline{d}_j = 0$, then, in practice, $\overline{A}_{i,j} = 0$ for $i : d_{j_i} > 0$. If so, none of these smaller basic variables bounds θ_j. A large step length then becomes likely.

The previous pivot rule attempts to estimate the step length associated with x_j based on the value of $\overline{d}_j$. Many other functions of the basic variables besides the piecewise linear one of (3.2) and (3.3) may yield helpful information about θ_j. How does one determine useful functions? Theorem 1 provides insight into this question by establishing a direct relation between $\overline{d}_j$ and θ_j for a suitable choice of f.

Theorem 1. *Assume $x_{j_i} > 0$, and set $d_N = 0$ and $d_{j_i} = 1/x_{j_i}$ for $i = 1, \ldots, m$. For some $j \in N$, let*

$$r = \underset{i:\overline{A}_{i,j}>0}{\operatorname{argmin}} \frac{x_{j_i}}{\overline{A}_{i,j}}.$$

Then,

$$\overline{d}_j = \frac{1}{\theta_j} + \sum_{\substack{i:\overline{A}_{i,j}\neq 0 \\ i\neq r}} \frac{\overline{A}_{i,j}}{x_{j_i}}. \tag{3.4}$$

Proof. In order to prove the theorem we show that θ_j^{-1} is a term of the summation that comprises $\overline{d}_j$. Note that r identifies the component of the basis indexing the variable that would depart the basis if x_j was chosen as entering variable. With this in mind,

$$\begin{aligned} \theta_j &= \min_{i:\overline{A}_{i,j}>0} \frac{x_{j_i}}{\overline{A}_{i,j}} = \frac{x_{j_r}}{\overline{A}_{r,j}} \\ \Rightarrow \frac{1}{\theta_j} &= \frac{\overline{A}_{r,j}}{x_{j_r}}. \end{aligned} \tag{3.5}$$

Now, compute $\overline{d}_j$ by its definition and extract θ_j^{-1} from the resulting summation:

$$\begin{aligned} \overline{d}_j = d_B^T \overline{A}_{\cdot j} &= \sum_{i=1}^{m} \frac{\overline{A}_{i,j}}{x_{j_i}} \\ &= \sum_{i:\overline{A}_{i,j}\neq 0} \frac{\overline{A}_{i,j}}{x_{j_i}} \\ &= \frac{\overline{A}_{r,j}}{x_{j_r}} + \sum_{\substack{i:\overline{A}_{i,j}\neq 0 \\ i\neq r}} \frac{\overline{A}_{i,j}}{x_{j_i}} \\ &= \frac{1}{\theta_j} + \sum_{\substack{i:\overline{A}_{i,j}\neq 0 \\ i\neq r}} \frac{\overline{A}_{i,j}}{x_{j_i}}. \end{aligned}$$

The last equality follows from (3.5). •

Notice that (3.5) implies that

$$\frac{1}{\theta_j} \geq \frac{\overline{A}_{i,j}}{x_{j_i}} \qquad i = 1, \ldots, m.$$

Therefore, the reciprocal of θ_j contributes the largest positive element to the sum comprising $\overline{d}_j$. One can use $\overline{d}_j$ to estimate θ_j. Smaller values of θ_j imply larger values of its reciprocal. Once again, negative values of $\overline{d}_j$ suggest relatively large step lengths. Clearly, the term

$$\gamma_j = \sum_{\substack{i:\overline{A}_{i,j} \neq 0 \\ i \neq r}} \frac{\overline{A}_{i,j}}{x_{j_i}}$$

may drastically distort this estimate. Nonetheless, in practice the canonical columns $\overline{A}_{\cdot j}$ tend to be fairly sparse, reducing the number of terms in the summation comprising γ_j. More importantly, $\overline{d}_j$ need only accurately estimate the size of θ_j relative to the step lengths associated with other potential entering variables. As long as the values of γ_j remain reasonably well behaved across all nonbasic variables, $\overline{d}_j$ will yield useful information about the relative sizes of the step lengths. In practice the assumption that $x_{j_i} > 0$ is unacceptable since virtually all practical problems exhibit degeneracy. In order to avoid this difficulty, let $\epsilon > 0$ represent a suitably small tolerance and set $d_{j_i} = \epsilon^{-1}$ if $x_{j_i} < \epsilon$. We can now formulate a pivot rule similar to (3.3):

$$\begin{aligned} \text{First Priority: } s &= \operatorname*{argmax}_{\overline{c}_j < 0, \overline{d}_j \leq 0} \overline{c}_j \overline{d}_j, \\ \text{Second Priority: } s &= \operatorname*{argmin}_{\overline{c}_j < 0, \overline{d}_j > 0} \frac{\overline{c}_j}{\overline{d}_j}. \end{aligned} \tag{3.6}$$

One can view this rule as an attempt to estimate cheaply the prohibitively expensive rule of maximizing the improvement in the objective function:

$$s = \operatorname*{argmin}_{j:\overline{c}_j < 0} \overline{c}_j \theta_j. \tag{3.7}$$

This procedure requires a solution of a system of equations and a ratio test for each nonbasic variable with negative reduced cost. Contrast this with (3.6), which requires only one additional solve. One could propose many other functions to define d_B. Regardless of the particular choice, Theorem 1 reveals the essential idea behind it. Explicit computation of θ_j for many nonbasic variables is hopelessly expensive (in a sequential computing environment), but the solution of a single system of linear equations can provide an inexpensive estimate of its value.

In [20] Kalan proposes a more elaborate version of (3.6) involving two extra pricing operations instead of one. Kalan's rule should provide more accurate information about selecting a good entering variable, but it also requires more computation time than (3.6). In [47] Todd motivates a pivot rule similar to (3.6) from the framework of an interior method for linear programming. To see the connection,

note that the components of d_B specified in Theorem 1 are precisely the components of the gradient of the logarithmic barrier function $\sum_{i=1}^{m} \ln x_{j_i}$.

3.4. Parametric Variants of the Simplex Method

The pivot rule (3.1) helps the simplex method avoid degenerate pivots. One expects this rule to perform well on highly degenerate problems. We now consider other pivot rules having nice properties with respect to degeneracy. The parametric simplex method proposed by Gass and Saaty (see [13]) provides a framework. Refer to Dantzig [7] for additional details. The parametric method does not select columns in order to avoid degenerate pivots, but it makes progress reducing dual infeasibility even when a decrease in the primal objective value is stalled by degeneracy. We consider a special case of the algorithm of Gass and Saaty. We provide a slightly different proof of convergence because of its applicability to an extension of the algorithm that incorporates dynamic pricing.

Consider a linear program of the form (1.1) with a parametric objective function $(c^T + \theta d^T)x$. Assume a feasible basis B_0; let N_0 index the corresponding nonbasic columns. Initialize the parametric cost row d as follows:

$$\begin{aligned} d_{B_0} &= 0, \\ d_j &= \|A_{\cdot j}\|_2 \qquad \text{for } j \in N_0. \end{aligned} \tag{3.8}$$

Actually, we only require that $d_j > 0$ for $j \in N_0$ to prove convergence. However, the particular choice (3.8) ensures that the forthcoming pivot rule remains invariant under column scaling. Associated with B_0 are the current values of the dual variables $\pi_0^T = c_{B_0}^T B_0^{-1}$ and the current reduced costs $\bar{c}_{N_0}^T = c_{N_0}^T - \pi_0^T A_{\cdot N_0}$. The algorithm initializes θ at a sufficiently large value so that the parametric objective function $\bar{c}_{N_0}(\theta) = \bar{c}_{N_0} + \theta \bar{d}_{N_0} > 0$. θ then decreases until it attains some value θ^1 where a component of $\bar{c}_{N_0}(\theta)$ attains zero. In other words, the current solution is optimal for the parameterized linear program provided that $\theta \geq \theta^1$. The component that equals zero identifies the entering basic variable. A ratio test defines this selection procedure. Assume no ties occur during this test. The usual simplex method ratio test then determines the outgoing basic variable; one can break ties arbitrarily. Pivot as usual, generating a new parametric objective function $\bar{c}_{N_1}(\theta) = \bar{c}_{N_1} + \bar{d}_{N_1}\theta$. To calculate the parametric reduced costs $\bar{d}_{N_1}$, observe that d is a second objective vector. Compute $\sigma_1^T = d_{B_1}^T B_1^{-1}$, and then set $\bar{d}_{N_1}^T = d_{N_1}^T - \sigma_1^T A_{\cdot N_1}$. Note that for the initial basis B_0, $\bar{d}_j = d_j$. We now repeat the pivot procedure. We shall see that, provided that there exists a unique choice of incoming variable, θ decreases strictly during each iteration. The parametric objective function $\bar{c}_N(\theta)$ remains nonnegative throughout the algorithm; the basis is optimal for the linear program (1.1) when θ attains zero.

Figure 3-1 summarizes the algorithm. Theorem 2 establishes convergence. Assume an optimal solution exists.

Theorem 2. *Provided there exists a unique choice of incoming variable during every iteration, the parametric algorithm of Figure 3-1 determines an optimal solution in a finite number of iterations.*

Proof. We use induction to show that the sequence of parametric values θ^0, θ^1, $\theta^2, \ldots$ generated by the algorithm decreases strictly during each iteration. This, combined with the fact that each basis corresponds to a unique value of θ, assures

Figure 3-1. Summary of the Parametric Algorithm

Given: An initial feasible basis $B = B_0$; $N = N_0$ indexes the corresponding non-basic columns.

1. Initialize parametric cost row d. Set $d_{B_0} = 0$ and $d_j = \|A_{\cdot j}\|_2$ for $j \in N_0$.
2. Set θ sufficiently large so that $\bar{c}_{N_0}(\theta) = \bar{c}_{N_0} + \bar{d}_{N_0}\theta > 0$. Note that $\bar{d}_j = d_j$ during the first iteration.

(Iterative Loop)

3. Decrease θ until

$$\exists\ s : \bar{c}_s + \theta\bar{d}_s = 0,\ \bar{c}_j + \theta\bar{d}_j > 0 \text{ for } j \in N/s. \tag{3.9}$$

Assume such an s exists. Use the following ratio test to determine s:

$$s = \underset{j:\bar{c}_j<0}{\operatorname{argmax}} -\frac{\bar{c}_j}{\bar{d}_j}. \tag{3.10}$$

Let θ be the corresponding maximum. If $\theta = 0$, go to 8.

4. Given x_s, determine the outgoing variable x_{j_r} by the usual simplex method ratio test. Ties may be broken arbitrarily. If the test reveals an unbounded solution, go to 9.
5. $A_{\cdot s}$ replaces $A_{\cdot j_r}$ as the r^{th} column of the basis. Update the current feasible solution as in the standard simplex method.
6. Calculate $\bar{c}$ and $\bar{d}$ for the new basis.
7. Go to 3.
8. Optimal solution found.
9. Terminate algorithm.

convergence. Consider the first iteration. Initially, $\theta = \theta^0$ is such that $\overline{c}_{N_0} + \theta \overline{d}_{N_0} > 0$. We then determine

$$\theta^1 = \max_{j:\overline{c}_j < 0} -\frac{\overline{c}_j}{\overline{d}_j}.$$

Since $\overline{d}_j = d_j = \|A_{\cdot j}\|_2 > 0$, it follows (see Figure 3-1) from (3.7) and (3.8) that $\theta^1 < \theta^0$. A strict decrease occurs during the first iteration; now consider iteration $k-1$. Let $\overline{c}^{k-1}$ and $\overline{d}^{k-1}$ represent the reduced costs of c and d at the start of iteration $k-1$. Similarly, let $\overline{a}_{ij}^{k-1}$ represent element (i,j) of the matrix $\overline{A}$ in the canonical form (1.2). θ^{k-1} represents the value of θ generated by the ratio test (3.8) during iteration $k-1$. By the induction hypothesis, $\theta^0 > \theta^1 > \ldots > \theta^{k-1}$, and

$$\begin{aligned} &\exists\, s: \ \overline{c}_s^{k-1} + \theta^{k-1}\overline{d}_s^{k-1} = 0, \\ &\qquad \overline{c}_j^{k-1} + \theta^{k-1}\overline{d}_j^{k-1} > 0, \ j \in N \setminus s. \end{aligned} \tag{3.11}$$

Note that x_s replaces x_{j_r} in the basis. The pivot element is then $\overline{a}_{r_s}^{k-1} > 0$. Perform the pivot and examine the resulting reduced costs for iteration k:

$$\begin{aligned} \overline{c}_j^k &= \overline{c}_j^{k-1} - \overline{a}_{rj}^{k-1}\frac{\overline{c}_s^{k-1}}{\overline{a}_{rs}^{k-1}}; \\ \overline{d}_j^k &= \overline{d}_j^{k-1} - \overline{a}_{rj}^{k-1}\frac{\overline{d}_s^{k-1}}{\overline{a}_{rs}^{k-1}}. \end{aligned}$$

Multiply $\overline{d}_j^k$ by θ^{k-1} and add to $\overline{c}_j^k$:

$$\begin{aligned} \overline{c}_j^k + \theta^{k-1}\overline{d}_j^k &= \overline{c}_j^{k-1} + \theta^{k-1}\overline{d}_j^{k-1} - \frac{\overline{a}_{rj}^{k-1}}{\overline{a}_{rs}^{k-1}} \underbrace{\left(\overline{c}_s^{k-1} + \theta^{k-1}\overline{d}_s^{k-1}\right)}_{=0 \text{ by } (3.11)} \\ &= \overline{c}_j^{k-1} + \theta^{k-1}\overline{d}_j^{k-1} > 0. \end{aligned}$$

The last inequality follows from (3.11). Since $\overline{c}_j^k + \theta^{k-1}\overline{d}_j^k > 0$ and $\theta^{k-1} > 0$, it follows that

$$\overline{c}_j^k < 0 \Rightarrow \overline{d}_j^k > 0. \tag{3.12}$$

Unless the current basis is optimal, (3.12) guarantees at least one potential pivot column satisfying (3.11). Also, observe that since $\overline{c}_j^k + \theta^{k-1}\overline{d}_j^k > 0$, a nonbasic column $\tilde{s}$ such that $\overline{c}_{\tilde{s}}^k > 0$ and $\overline{d}_{\tilde{s}}^k < 0$ cannot satisfy (3.11) for $\theta < \theta^{k-1}$. Hence, we only need consider $j \in N : \overline{c}_j < 0$ as candidates to index the entering variable. This validates the ratio test (3.10). We can now show that $\theta^k < \theta^{k-1}$. Since $\overline{c}_j^k + \theta^{k-1}\overline{d}_j^k > 0$, (3.12) implies the existence of θ^k and $\hat{s}$ such that $\theta^k < \theta^{k-1}$, $\overline{c}_{\hat{s}}^k + \theta^k \overline{d}_{\hat{s}}^k = 0$, and $\overline{c}_j^k + \theta^k \overline{d}_j^k > 0$. To determine θ^k and $\hat{s}$, we choose θ^k as the

$\overline{c}_j^k + \theta^k \overline{d}_j^k > 0$. To determine θ^k and $\hat{s}$, we choose θ^k as the smallest possible value that satisfies the condition

$$\begin{aligned}
&\overline{c}_j^k + \theta^k \overline{d}_j^k \geq 0 \quad \text{for } j:\ \overline{c}_j^k < 0,\ \overline{d}_j^k > 0 \\
&\Leftrightarrow \overline{c}_j^k \geq -\theta \overline{d}_j^k \quad \text{for } j:\ \overline{c}_j^k < 0 \\
&\Leftrightarrow \theta \geq -\frac{\overline{c}_j^k}{\overline{d}_j^k} \quad \text{for } j:\ \overline{c}_j^k < 0 \\
&\Leftrightarrow \theta = \max_{j:\overline{c}_j^k < 0} -\frac{\overline{c}_j^k}{\overline{d}_j^k}, \text{ and } \hat{s} = \operatorname*{argmax}_{j:\overline{c}_j^k < 0} -\frac{\overline{c}_j^k}{\overline{d}_j^k}.
\end{aligned}$$

Thus, the ratio test (3.10) determines $\hat{s}$ and $\theta^k < \theta^{k-1} < \ldots < \theta^1 < \theta^0$. Remember that we have assumed no ties occur in this test. A unique value of θ corresponds to each basis since the ratio test involves $\overline{c}$ and $\overline{d}$. Therefore, a basis cannot repeat itself during the algorithm. This completes the inductive proof. •

Theorem 2 assumes that no ties occur during the ratio test (3.10). A random perturbation of the initial values of d_{N_0} validates this assumption with probability one.

The values of θ generated during the parametric algorithm provide a measure of the level of dual infeasibility. Even when stalling (a long sequence of degenerate pivots) occurs in the primal, the algorithm progresses in the dual. This suggests that the method will perform well in the presence of degeneracy. The screening criterion (3.1) works in the primal; it tries to avoid degenerate pivots and, hence, stalling by using a second objective function d. The parametric algorithm also utilizes a second objective function, albeit a constant one. From the perspective of the primal, however, no features of this second objective appear to help it avoid degenerate pivots. As we shall see, (3.1) reduces the percentage of degenerate pivots, while the parametric algorithm does not. Nonetheless, computational tests in Chapter 4 reveal the effectiveness of both methods on highly degenerate problems.

The inability of the parametric algorithm to avoid degenerate pivots suggests the potential of a variant that can avoid degenerate pivots while still decreasing θ during each iteration. Unfortunately, the parametric algorithm lacks the freedom to choose the incoming variable. θ need not decrease unless the ratio test (3.10) determines the incoming variable. However, provided that one initializes $d_j > 0$ for $j \in N_0$, θ decreases monotonically under the parametric pivot rule. Since the algorithm generates a sequence of feasible bases, one can reinitialize the parametric objective at any iteration. We shall use this fact to formulate a modified parametric algorithm that screens for degenerate pivots without sacrificing the monotonic decrease of θ.

Let us begin by defining some additional notation. Let d^1 and d^2 represent the parametric and dynamic objectives, respectively. Let $\overline{d}^1$ and $\overline{d}^2$ be the corresponding reduced costs. Consider any iteration. As before, s indexes the entering variable chosen by the parametric algorithm:

$$s = \operatorname*{argmax}_{j:\overline{c}_j < 0} -\frac{\overline{c}_j}{\overline{d}_j^1}. \tag{3.13}$$

Again, assume s is unique. If $\overline{d}_s^2 > 0$, Lemma 3 implies the resulting pivot will be degenerate. Let q index the variable that maximizes the ratio in (3.13) while also

passing the screening criterion of (3.1):

$$q = \underset{j:\bar{c}_j<0,\bar{d}_j^2\leq 0}{\text{argmax}} \; -\frac{\bar{c}_j}{\bar{d}_j^1}. \tag{3.14}$$

Assume that during all previous iterations $q = s$. In other words, the entering variable was always a valid choice under the parametric algorithm; hence θ decreased during each iteration. Suppose at the current iteration $q \neq s$. Then, since q does not maximize the ratio in (3.13),

$$-\frac{\bar{c}_s}{\bar{d}_s^1} > -\frac{\bar{c}_q}{\bar{d}_q^1}. \tag{3.15}$$

We wish to reinitialize the parametric objective vector so that q indexes a legitimate entering column with respect to the parametric algorithm. In order to do so, determine $\delta > 0$ such that

$$-\frac{\bar{c}_s}{\bar{d}_s^1} < -\frac{\bar{c}_q}{\bar{d}_q^1 - \delta}.$$

The following lemmas motivate the proper selection of δ.

Lemma 5.

$$\frac{\bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1}{\bar{c}_s} < \bar{d}_q^1.$$

Proof. Suppose the contrary:

$$\frac{\bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1}{\bar{c}_s} \geq \bar{d}_q^1.$$

Remember that all previous pivots were valid under the parametric algorithm. All properties of the algorithm remain true, so by (3.12), $\bar{c}_s < 0$ and $\bar{d}_s^1 > 0$. Subtracting $\bar{d}_q^1$ from both sides of the last inequality,

$$\begin{aligned} \frac{\bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1 - \bar{c}_s\bar{d}_q^1}{\bar{c}_s} &\geq 0 \\ \Rightarrow -\frac{\bar{c}_q\bar{d}_s^1}{\bar{c}_s} &\geq 0 \\ \Rightarrow -\bar{c}_q\bar{d}_s^1 &\leq 0. \end{aligned}$$

But $\bar{d}_s^1 > 0$ and $\bar{c}_q < 0$, establishing a contradiction. •

Lemma 6.

$$\frac{\bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1}{\bar{c}_s} > 0.$$

Proof. Since $s \neq q$, use (3.15) along with the fact that $\bar{d}_q^1$ and $\bar{d}_s^1$ are positive:

$$\begin{aligned} -\frac{\bar{c}_s}{\bar{d}_s^1} &> -\frac{\bar{c}_q}{\bar{d}_q^1} \\ \Rightarrow \bar{c}_s\bar{d}_q^1 &< \bar{c}_q\bar{d}_s^1 \\ \Rightarrow \bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1 &< 0 \qquad \text{(recall that } \bar{c}_s < 0\text{)} \\ \Rightarrow \frac{\bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1}{\bar{c}_s} &> 0. \quad \bullet \end{aligned}$$

Lemma 7.

$$\begin{aligned} &\exists\ \delta > \frac{\bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1}{\bar{c}_s} \text{ such that} \\ &-\frac{\bar{c}_s}{\bar{d}_s^1} < -\frac{\bar{c}_q}{\bar{d}_q^1 - \delta}. \end{aligned} \tag{3.16}$$

Proof. Lemma 5 and property (3.12) provide the proof. By Lemma 5,

$$\frac{\bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1}{\bar{c}_s} < \bar{d}_q^1.$$

Therefore,

$$\exists\ \delta > \frac{\bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1}{\bar{c}_s} \text{ such that } \bar{d}_q^1 > \delta. \tag{3.17}$$

Note that $\bar{d}_q^1 - \delta > 0$. Multiplying (3.17) by $\bar{c}_s < 0$ reveals that

$$\begin{aligned} \bar{c}_s\delta &< \bar{c}_s\bar{d}_q^1 - \bar{c}_q\bar{d}_s^1 \\ \Rightarrow -\bar{c}_s(\bar{d}_q^1 - \delta) &< -\bar{c}_q\bar{d}_s^1 \quad \text{(recall that } \bar{d}_s^1 > 0\text{)} \\ \Rightarrow -\frac{\bar{c}_s}{\bar{d}_s^1} &< -\frac{\bar{c}_q}{\bar{d}_q^1 - \delta}. \quad \bullet \end{aligned}$$

Given these lemmas, Theorem 3 shows how to reinitialize the parametric objective vector so that x_q enters the basis instead of x_s.

Theorem 3. *Assume the standard parametric ratio test (3.13) determines a unique entering variable during the first $k-1$ iterations of the parametric algorithm. Suppose that $q \neq s$ at iteration k, and the parametric objective vector d^1 is replaced by the following vector d^3:*

$$\begin{aligned} d^3_B &= 0 \\ d^3_q &= \overline{d}^1_q - \delta \\ d^3_j &= \overline{d}^1_j \quad \text{for } j \in N\text{: } \overline{c}_j < 0,\ j \neq q \\ d^3_j &= d^1_j \quad \text{for } j \in N\text{: } \overline{c}_j \geq 0. \end{aligned} \tag{3.18}$$

Then $\exists\ \delta$ such that q indexes a valid choice of pivot column for the parametric algorithm.

Proof. To prove the theorem, we must show:

(i) that d^3 is a legitimate choice of initial vector for the standard parametric algorithm.

(ii) that the algorithm will select x_q to enter the basis, i.e.

$$q = \underset{j:\overline{c}_j<0}{\operatorname{argmax}} -\frac{\overline{c}_j}{\overline{d}^3_j}.$$

(iii) that the value of the parameter $\theta^k < \theta^{k-1}$.

Let B and N index the basic and nonbasic variables of iteration k. Consider d^3. By Lemmas 6 and 7, $\exists\ \delta > 0$ satisfying (3.16) such that $d^3_q = \overline{d}^1_q - \delta > 0$. By property (3.12) of the parametric algorithm, $\overline{d}^1_j > 0$ for $j:\ \overline{c}_j < 0$. Furthermore, $d^1_j = \|A_{\cdot j}\|_2 > 0$. Hence $d^3_j > 0$ for $j \in N$, so d^3 is a legitimate initial vector for the parametric algorithm. This establishes (i).

Now consider the new parametric reduced costs $\overline{d}^3$. Since $d^3_B = 0$, $\sigma^T_3 = (d^3_B)^T B^{-1} = 0$, so $\overline{d}^3_j = d^3_j - \sigma^T_3 A_{\cdot j} = d^3_j$. Using Lemma 7 and (3.18),

$$-\frac{\overline{c}_q}{\overline{d}^3_q} = -\frac{\overline{c}_q}{\overline{d}^1_q - \delta} > -\frac{\overline{c}_s}{\overline{d}^1_s} = -\frac{\overline{c}_s}{\overline{d}^3_s}.$$

Also, since s maximized the ratio for the standard parametric pivot rule (3.13),

$$-\frac{\overline{c}_s}{\overline{d}^1_s} > -\frac{\overline{c}_j}{\overline{d}^1_j} = -\frac{\overline{c}_j}{\overline{d}^3_j} \quad \text{for } j:\overline{c}_j < 0,\ j \neq q.$$

Combining inequalities,

$$-\frac{\overline{c}_q}{\overline{d}^3_q} > -\frac{\overline{c}_j}{\overline{d}^3_j} \quad \text{for } j:\overline{c}_j < 0,\ j \neq q.$$

In other words,

$$q = \underset{j:\bar{c}_j<0}{\operatorname{argmax}} -\frac{\bar{c}_j}{\bar{d}^3_j}.$$

This establishes (*ii*).

All that remains is to show that the parameter decreases when x_q enters the basis. Let θ^k_s represent the value of the parameter at iteration k determined by the normal parametric pivot rule (3.13). In other words,

$$\theta^k_s = -\frac{\bar{c}_s}{\bar{d}^1_s}.$$

Similarly, define θ^k_q as the parameter associated with entering x_q into the basis after reinitializing the parametric objective row:

$$\theta^k_q = \frac{-\bar{c}_q}{\bar{d}^1_q - \delta}.$$

Using Lemma 7, we see that

$$\delta = \frac{\bar{c}_s\bar{d}^1_q - \bar{c}_q\bar{d}^1_s}{\bar{c}_s} + \epsilon,$$

where $\epsilon \in (0, \bar{\epsilon})$ for some $\bar{\epsilon} > 0$. Substituting the last equation into the previous one,

$$\begin{aligned}\theta^k_q &= \frac{-\bar{c}_q}{\bar{d}^1_q - \left((\bar{c}_s\bar{d}^1_q - \bar{c}_q\bar{d}^1_s)/\bar{c}_s\right) - \epsilon} \\ &= \frac{-\bar{c}_q}{(\bar{c}_q\bar{d}^1_s - \epsilon\bar{c}_s)/\bar{c}_s} = \frac{-\bar{c}_s\bar{c}_q}{\bar{c}_q\left(\bar{d}^1_s - \epsilon\bar{c}_s/\bar{c}_q\right)} \\ &= \frac{-\bar{c}_s}{\underbrace{\bar{d}^1_s}_{>0} - \underbrace{\epsilon\bar{c}_s/\bar{c}_q}_{>0}} > \theta^k_s.\end{aligned}$$

The last inequality holds since $\theta^k_s = -\bar{c}_s/\bar{d}^1_s$. Although $\theta^k_q > \theta^k_s = -\bar{c}_s/\bar{d}^1_s$, note that $(\theta^k_q - \theta^k_s)$ can be made arbitrarily small with a suitable choice of ϵ. Recall that $\theta^k_s < \theta^{k-1}$ because all previous iterations have obeyed the normal parametric pivot rule. It follows that $\exists\ \epsilon$ sufficiently small so that $\theta^k_q < \theta^{k-1}$. Thus, the parameter decreases by a positive amount, proving (iii). We have established that, with a suitable choice of ϵ, q is the proper column selection for the parametric algorithm with parametric objective function d^3. •

Given the results of Theorem 2, Theorem 3 shows that the modified parametric algorithm summarized in Figure 3-2 terminates in a finite number of iterations

Figure 3-2. Summary of a Modified Parametric Algorithm

Given: An initial feasible basis $B = B_0$; $N = N_0$ indexes the corresponding non-basic columns.

1. Initialize parametric cost row d^1. Set $d^1_{B_0} = 0$ and $d_j = \|A_{\cdot j}\|_2$ for $j \in N_0$.
2. Set θ sufficiently large so that $\overline{c}_{N_0}(\theta) = \overline{c}_{N_0} + \overline{d}_{N_0}\theta > 0$.

(Iterative Loop)

3. Determine s and q by the ratio tests (3.13) and (3.14). If $\theta = 0$, go to 12.
4. If $s = q$, go to 9.
5. Since $s \neq q$, reinitialize d^1. Determine $\overline{\epsilon} > 0$ such that

$$\frac{-\overline{c}_s}{\overline{d}^1_s - \overline{\epsilon c}_s/\overline{c}_q} < \theta^{k-1}.$$

6. Set

$$\delta = \frac{\overline{c}_s\overline{d}^1_q - \overline{c}_q\overline{d}^1_s}{\overline{c}_s} + \epsilon$$

for $\epsilon \in (0, \overline{\epsilon})$ so that

$$\overline{d}^1_q > \delta > \frac{\overline{c}_s\overline{d}^1_q - \overline{c}_q\overline{d}^1_s}{\overline{c}_s}.$$

7. Reinitialize d^1 as follows:

$$\begin{aligned} d^1_B &\leftarrow 0 \\ d^1_q &\leftarrow \overline{d}^1_q - \delta \\ d^1_j &\leftarrow \overline{d}^1_j \quad \text{for } j : \overline{c}_j < 0,\ j \neq q \\ d^1_j &\leftarrow d^1_j \quad \text{for } j : \overline{c}_j \geq 0. \end{aligned}$$

8. Variable x_q enters the basis. Go to 10.
9. Variable x_s enters the basis.
10. Determine the pivot row by the standard simplex method ratio test. If the test reveals an unbounded solution, go to 13.
11. Pivot and update the basis. Update the current feasible solution. Compute $\overline{c}$, $\overline{d}^1$, and $\overline{d}^2$ for the new basis. Go to 3.
12. Optimal solution found.
13. Terminate algorithm.

if accompanied by a suitable degeneracy resolution technique. The algorithm initializes the parametric objective vector in the normal way and proceeds with the standard parametric method until a pivot selection s fails the screening criterion (3.1). At this point a feasible basis exists, so we reinitialize the parametric vector so that x_q enters the basis instead of x_s. Lemmas 5–7 and Theorem 3 describe the reinitialization procedure and show that the parameter still decreases.

Note that the modified algorithm requires a degeneracy resolution technique to guarantee convergence while the original one does not. This distinction arises because the modified algorithm alters the parametric objective function, whereas the parametric objective of the original one remains unchanged. It therefore becomes conceivable that a basis could repeat itself where the parametric objective has doubled and the parameter has halved (see [48]). In this instance the parameter would decrease at each iteration yet never attain zero. If accompanied by a degeneracy resolver, the modified algorithm will terminate in a finite number of iterations since it always selects variables with negative reduced costs to enter the basis.

Each iteration of the modified parametric algorithm requires computation of two extra vectors of reduced costs. Also, the reinitialization procedure (3.18) involves some more work. The total additional computation exceeds that of any of the previously described pivot rules. Nonetheless, the approach incorporates the beneficial characteristics of the parametric method and pivot rule (3.1). It progresses in the dual during stalls in the primal, but it also attempts to avoid such stalls.

3.5. Reduction of the Additional Computation

All of the previously described pivot rules compute reduced costs on a second objective vector d. In the context of the revised simplex method, one must determine d_B, solve an extra system of equations

$$\sigma^T B = d_B^T, \tag{3.19}$$

and then calculate reduced costs $\bar{d}_j$ for $j : \bar{c}_j < 0$. Although not prohibitively expensive, these steps comprise a significant fraction of the time required for a simplex method iteration. In this section we explore techniques to reduce the additional work.

An opportunity to save time arises during the solution of the system of equations (3.19). Notice the similarity between solving for σ and solving for the dual variables π:

$$\pi^T B = c_B^T. \tag{3.20}$$

One should solve (3.19) and (3.20) simultaneously. One could call a subroutine twice:

CALL SOLVE($\pi, c_B, \ldots$)

<additional code>

CALL SOLVE($\sigma, d_B, \ldots$).

Assuming an LU factorization represents B, each call involves solving the linear system $w^T B = z^T$, which in turn requires solving the linear systems

$$y^T U = z^T$$

$$w^T L = y^T.$$

Thus each subroutine call must access the array containing the nonzeros of the lower triangular matrix L and the upper triangular matrix U. Instead, suppose one modifies the subroutine so it computes π and σ during the same call:

$$\text{CALL SOLVE}(\pi, \sigma, c_B, d_B, \ldots).$$

This approach requires only one access of the array containing the factorization of the basis B and may therefore reduce the computation time involved.

In certain situations, the extra solve (3.19) becomes unnecessary. In particular we will demonstrate how to update σ for the pivot rules (3.1), (3.2), (3.3), and (3.6) provided that a degenerate pivot occurred during the previous iteration. We shall also see that one can always skip the extra solve for a parametric objective vector. Again, e_r denotes a unit column vector with a one in the r^{th} component.

Theorem 4. Let B_k, π_k, and σ_k represent the basis, dual variables, and second objective multipliers during iteration k. Let $\bar{a}_{rs}$, $\bar{c}_j$, and $\bar{d}_j$ be the pivot element, reduced costs and second objective reduced costs. Suppose that the second objective vector d changes by only one component after iteration k:

$$d_{B_{k+1}} = d_{B_k} + \rho e_r, \ \rho \in R^1. \tag{3.21}$$

Then

$$\sigma_{k+1}^T = \sigma_k^T + \frac{\rho - \sigma_k^T(A_{.s} - A_{.j_r})}{\bar{c}_s}(\pi_{k+1}^T - \pi_k^T). \tag{3.22}$$

Proof. To prove the theorem we exploit the similarity between the linear systems (3.19) and (3.20) when (3.21) holds. Note that $\bar{c}_s < 0$ and $\bar{a}_{rs} > 0$, so all of the quotients formed in the proof remain well defined.

Let s and j_r index the incoming and outgoing basic variables, respectively, during iteration k. Then,

$$B_{k+1} = B_k + (A_{.s} - A_{.j_r})e_r^T. \tag{3.23}$$

Let $u = (A_{.s} - A_{.j_r})$, and suppose v solves the linear system

$$v^T B_k = e_r^T. \tag{3.24}$$

In other words, v^T contains the r^{th} row of B_k^{-1}. Observe from (3.23) and (3.24) that

$$B_{k+1} = (I + uv^T)B_k. \tag{3.25}$$

Since $\sigma_{k+1}^T B_{k+1} = d_{B_{k+1}}^T$ and $d_{B_{k+1}} = d_{B_k} + \rho e_r$, substituting for B_{k+1} as in (3.25) implies that

$$\begin{aligned} \sigma_{k+1}^T(I + uv^T)B_k &= d_{B_k}^T + \rho e_r^T \\ &\overset{\text{by (3.19)}}{=} \overbrace{\sigma_k^T B_k}^{\text{by (3.19)}} + \rho \overbrace{v^T B_k}^{\text{by (3.24)}} \\ \Rightarrow \sigma_{k+1}^T(I + uv^T) &= \sigma_k^T + \rho v^T. \end{aligned} \tag{3.26}$$

Now, proceed similarly to derive an analogous expression for π_{k+1}. First of all, note that

$$c_{B_{k+1}} = c_{B_k} + (c_s - c_{j_r})e_r.$$

Since $\pi_{k+1}^T B_{k+1} = c_{B_{k+1}}^T$ it follows from (3.25) that

$$\begin{aligned}
\pi_{k+1}^T(I + uv^T)B_k &= c_{B_k}^T + (c_s - c_{j_r})e_r^T \\
&= \pi_k^T B_k + (c_s - c_{j_r})v^T B_k \\
\Rightarrow \pi_{k+1}^T(I + uv^T) &= \pi_k^T + (c_s - c_{j_r})v^T. \qquad (3.27)
\end{aligned}$$

Rearranging (3.26) and (3.27),

$$\sigma_{k+1}^T - \sigma_k^T = (\rho - \sigma_{k+1}^T u)v^T, \qquad (3.28)$$

$$\pi_{k+1}^T - \pi_k^T = ((c_s - c_{j_r}) - \pi_{k+1}^T u)v^T. \qquad (3.29)$$

Substituting (3.29) into (3.28),

$$\sigma_{k+1}^T - \sigma_k^T = \frac{(\rho - \sigma_{k+1}^T u)}{((c_s - c_{j_r}) - \pi_{k+1}^T u)}(\pi_{k+1}^T - \pi_k^T). \qquad (3.30)$$

We shall later verify that the denominator in this expression cannot equal zero. We must now derive expressions for $\sigma_{k+1}^T u$ and $\pi_{k+1}^T u$. Begin by multiplying both sides of (3.28) by u:

$$\sigma_{k+1}^T u - \sigma_k^T u = (\rho - \sigma_{k+1}^T u)v^T u.$$

Rearranging,

$$\begin{aligned}
\sigma_{k+1}^T(u + uv^T u) &= \rho v^T u + \sigma_k^T u \\
\Rightarrow \sigma_{k+1}^T u(1 + v^T u) &= \rho v^T u + \sigma_k^T u \\
\Rightarrow \sigma_{k+1}^T u &= \frac{\rho v^T u + \sigma_k^T u}{(1 + v^T u)}. \qquad (3.31)
\end{aligned}$$

Note that $v^T u = e_r^T B_k^{-1}(A_{\cdot s} - A_{\cdot j_r}) = \bar{a}_{rs} - 1$, so the denominator in (3.31) is nonzero. Proceed similarly to derive $\pi_{k+1}^T u$:

$$\begin{aligned}
\pi_{k+1}^T u - \pi_k^T u &= ((c_s - c_{j_r}) - \pi_{k+1}^T u)v^T u \\
\Rightarrow \pi_{k+1}^T u(1 + v^T u) &= (c_s - c_{j_r})v^T u + \pi_k^T u \\
\Rightarrow \pi_{k+1}^T u &= \frac{(c_s - c_{j_r})v^T u + \pi_k^T u}{(1 + v^T u)}. \qquad (3.32)
\end{aligned}$$

Consider the denominator in (3.30). Remember that s and j_r index nonbasic and basic variables respectively at the start of iteration k. Substituting the value of $\pi_{k+1}^T u$ from (3.32) and regrouping under a common denominator,

$$\begin{aligned}
(c_s - c_{j_r}) - \pi_{k+1}^T u &= \frac{\overbrace{\left((c_s - c_{j_r})(1 + v^T u) - (c_s - c_{j_r})v^T u\right)}^{simplify} - \pi_k^T u}{(1 + v^T u)} \\
&= \frac{(c_s - c_{j_r}) - \pi_k^T \overbrace{(A_{\cdot s} - A_{\cdot j_r})}^{u}}{(1 + v^T u)} \\
&= \frac{\overbrace{(c_s - \pi_k^T A_{\cdot s})}^{\bar{c}_s} - \overbrace{(c_{j_r} - \pi_k^T A_{\cdot j_r})}^{0}}{(1 + v^T u)} \\
&= \frac{\bar{c}_s}{(1 + v^T u)}. \qquad (3.33)
\end{aligned}$$

Thus, the denominator is indeed nonzero. Apply the same logic to the numerator of (3.30) by using (3.31):

$$\begin{aligned}\rho - \sigma_{k+1}^T u &= \frac{\rho(1+v^T u) - (\rho v^T u + \sigma_k^T u)}{(1+v^T u)} \\ &= \frac{\rho - \sigma_k^T u}{(1+v^T u)}. \end{aligned} \tag{3.34}$$

Substituting (3.33) and (3.34) into (3.30) and rearranging,

$$\sigma_{k+1}^T = \sigma_k^T + \frac{\rho - \sigma_k^T (A_{\cdot s} - A_{\cdot j_r})}{\overline{c}_s}(\pi_{k+1}^T - \pi_k^T). \quad \bullet$$

The implications of Theorem 4 depend on the particular choice of second objective. For any of the pivot rules utilizing dynamic pricing, the result reduces computation time whenever a degenerate pivot occurs. In this case the values of the basic variables do not change between successive iterations; the simplex method merely exchanges two variables that equal zero. Therefore, $d_{B_{k+1}} = d_{B_k}$, which implies that $\rho = 0$. Note that $\sigma_k^T A_{\cdot s} = \overline{d}_s$, and, since $\sigma^T B = d_B^T$, $\sigma_k^T A_{\cdot j_r} = d_{j_r}$. The result of Theorem 4 simplifies to

$$\sigma_{k+1}^T = \sigma_k^T - \frac{(\overline{d}_s - d_{j_r})}{\overline{c}_s}(\pi_{k+1}^T - \pi_k^T).$$

In order to perform the computational tests, it was necessary to handle bounded variables. In this case a degenerate pivot may occur when the value of the incoming basic variable differs from that of the outgoing one. Hence, $d_{B_{k+1}} = d_{B_k} + \rho e_r$, where $\rho \neq 0$; Theorem 4 still holds. For details about the different values of ρ generated by the various types of degenerate pivots, refer to the appendix.

Let us now consider parametric algorithms. In this case

$$\begin{aligned}\sigma_k^T u &= \sigma_k^T A_{\cdot s} - \sigma_k^T A_{\cdot j_r} \\ &= \underbrace{(-d_s + \sigma_k^T A_{\cdot s})}_{-\overline{d}_s} + \underbrace{(d_{j_r} - \sigma_k^T A_{\cdot j_r})}_{0} + d_s - d_{j_r} \\ &= d_s - (\overline{d}_s + d_{j_r}).\end{aligned}$$

Also, regardless of whether a degenerate pivot occurs,

$$d_{B_{k+1}} = d_{B_k} + (d_s - d_{j_r})e_r,$$

so $\rho = d_s - d_{j_r}$. Substituting for ρ and $\sigma_k^T u$ into the result of Theorem 4 yields

$$\sigma_{k+1}^T = \sigma_k^T + \frac{\overline{d}_s}{\overline{c}_s}(\pi_{k+1}^T - \pi_k^T).$$

Therefore, despite the existence of the second objective, one never need perform an extra solve during the parametric algorithm. Instead we merely update the vector σ. The same conclusion applies to any variant of the simplex method that utilizes two constant objective vectors and computes a pair of reduced costs. One could also avoid the extra solve by maintaining a pair of vectors for the reduced costs, but that approach requires extra storage and extra array references, and it is not amenable to partial pricing.

CHAPTER 4: COMPUTATIONAL RESULTS

4.1. Preliminaries

The author tested the previously discussed pivot rules on a set of 62 practical problems, 53 of which are publicly available. Problem sizes (excluding slacks) range from small (28×32) to large (2263×9799). The simplex method solved some of these problems quite efficiently but had great difficulty with others. A heuristic measure of its performance is the ratio of iterations required to the number of rows in the constraints. If this ratio exceeds 5.0, one can consider the problem difficult.

The problems were partitioned into different sets in an attempt to distinguish certain characteristics. The KETRON set, the only proprietary problems tested, consists of nine highly degenerate problems. Degenerate pivots occurred during at least 30 percent of the iterations for each problem when solved by the standard simplex method; sometimes the percentage exceeded 80. The PILOT set contains four linear programs generated by variants of the PILOT model. A large-scale economic model, PILOT uses various units of measurement of the activity levels and input-output items between the many different sectors of the economy. These conversions of units have resulted in notoriously poor scaling of the constraints. Although all four problems arise from the same model, examination of the structure of each problem (see [27]) reveals substantial differences. A collection of 14 staircase problems comprises the STAIRCASE set. The STANFORD set consists of the 12 problems used in Chapter 2 to test the modified feasible direction method. The fifth group, labeled the SHIP set, contains six related problems. Unfortunately, the author has no details on them. The remaining 17 problems form the MISCELLANEOUS set. In general, these problems lack any known categorizable features.

As with the tests of Chapter 2, the author modified MINOS 5.1 in order to implement the desired pivot rules. The primary changes occurred in the pricing routine that determines the incoming column. All other aspects of the simplex method remained unchanged. Once again, one can attribute distinctions in performance to differences in the pivot rules, not inconsistencies in the implementations.

Figure 4-1 lists the test sets and problem sizes. MINOS contains an option that scales a problem before commencing the simplex method. Since scaling may drastically alter performance, both scaled and unscaled problems were tested. Partial pricing was not used.

4.2. Screening for Degenerate Pivots

We begin with the results for pivot rule (3.1). Recall that this rule screens columns in an attempt to avoid degenerate pivots. We therefore label the rule as the "Degeneracy Screen" in the following figures. Figure 4-2 displays the results for unscaled problems. We compare both iterations and CPU time with the standard simplex method of MINOS 5.1. Note that MINOS 5.1 contains an anti-cycling procedure designed to both prevent cycling and improve performance. Iteration counts reveal if the information provided by the second objective function is useful; the times determine if that information is worth the extra work. Also shown is the geometric mean of the resulting ratios for each test set; it appears in the boxes directly below each test set. Assuming all problems are equally important (perhaps an unrealistic assumption given the disparity in the size and difficulty of each problem), it measures relative performance of the new pivot rule for each test set. Values less than 1.0 imply superior performance of the new rule.

The Degeneracy Screen performed quite well on the KETRON set. This is not surprising, given that the rule is explicitly designed to avoid degenerate pivots. It substantially reduced both iterations and time on most of these problems. It performed exceptionally well on the larger, more difficult problems. The new pivot rule also succeeded on three of the four problems in the PILOT set. PILOTS, the unsuccessful problem, causes only 3 percent degenerate pivots when solved by the standard simplex method. It is therefore not surprising that, with respect to time, the Degeneracy Screen failed to outperform the regular pivot rule on that problem. The new pivot rule consistently reduced the iteration counts on the STAIRCASE set as well. However, on many problems, the decrease in iterations didn't quite compensate for the extra work per iteration. On the basis of CPU time the two pivot rules performed similarly on this set. Slightly worse results occurred with the STANFORD set, as only a slight overall reduction in iterations occurred. The STANFORD set includes ISRAEL, which caused the most trouble for the new rule. Test results on the SHIP set were consistently unfavorable with respect to both iterations and time. One should note, however, that, given the problem sizes, the standard pivot rule performs extremely well on these problems, so pivot rule (3.1) also solves them quite efficiently. (3.1) consistently decreases the iteration counts of the MISCELLANEOUS set, but it doesn't quite break even in terms of time. We again encounter many problems where time increases despite a decrease in iterations. Notice that it performs extremely well on the problem FFFFF800.

Since the Degeneracy Screen tries to avoid degenerate pivots, it is interesting to examine the level of degeneracy in each test problem. Figure 4-3 contains information on the frequency of degenerate pivots for each pivot rule on unscaled problems. The new pivot rule typically reduces the frequency of such blocked pivots, sometimes quite substantially (see NZFRI, PILOTJA, and FFFFF800). This suggests that part of its success derives from the ability to avoid unnecessary pivots. On highly degenerate problems, it seems likely that traversing any path of vertices to an optimal solution will require some degenerate pivots. Indeed, many practical problems contain blocks of activities for which dropping a key activity of a block implies dropping all other activities in the block. Nonetheless, Figure 4-3 suggests that many such pivots performed by the simplex method are unnecessary. Notice also that the pivot rule can still work well even when it doesn't reduce the percentage of blocked pivots; TUFF, CYCLE and WOODW provide examples of this behavior.

Figure 4-4 contains comparisons of the same two pivot rules for scaled problems. Relative performance remains virtually unchanged for each test set. Very few individual problems show significant differences; SCSD8 and FFFFF800 provide two exceptions. The PILOT set is particularly noteworthy, since scaling dramatically improves the performance of the standard pivot rule. Nonetheless, the Degeneracy Screen still results in significant improvement when applied to the scaled problems. As before, it typically reduces the frequency of blocked pivots; refer to Figure 4-5 for details.

4.3. Screening for Small Step Lengths

Figure 4-6 contains results for pivot rule (3.2) applied to unscaled problems. Recall that this rule attempts to exclude potential incoming variables that would result in small step lengths (hence the label "Small Screen" in Figures 4-6 and 4-7). It performs fairly well on the KETRON set, yielding a moderate overall improvement in times. It does not perform as well on these highly degenerate problems as the Degeneracy Screen. This is to be expected since this rule sacrifices the ability to avoid zero step lengths in order to gain additional information about positive ones.

However, this rule works extremely well on the PILOT set, achieving tremendous reductions of both iterations and time on three of the four problems. On PILOTS, the fourth problem, it reduces iterations but marginally increases time. This is more than offset by its performance on the other three problems, particularly PILOTJA. The approach does not work as well on the STAIRCASE and STANFORD sets, as it marginally reduces iterations but typically increases time. The SHIP set once again proves difficult, although the new pivot rule still performs well given the problem sizes. Iterations for the MISCELLANEOUS set typically decrease, but these reductions frequently fail to compensate for the extra computation. Recall that pivot rules (3.2) and (3.3) involve computation of the mean of the basic variables. This comprises a significant portion of the extra work, especially with respect to the bounded variable format of MINOS. Given the experimental nature of these tests, the author computed the exact value of the mean during each iteration. In practice, one could approximate the mean using a variety of computationally cheaper approaches, thus reducing the times significantly.

Figure 4-7 contains results for pivot rule (3.2) on scaled problems. On most sets scaling marginally improves its performance relative to the usual pivot rule. On the PILOT set, relative performance declines substantially, primarily because scaling improves the standard rule so dramatically. On average the new pivot rule still reduces iterations, but it now causes a significant increase in time for two of the four problems.

4.4. Piecewise Linear Estimation of the Step Length

Figure 4-8 displays results for pivot rule (3.3) on unscaled problems. Recall that (3.3) is a three-priority pivot rule that uses a piecewise linear function to estimate the step length associated with a potential incoming variable. We therefore use the abbreviation "PLSE" in Figures 4-8 and 4-9. The approach generally succeeds on the KETRON set, except for the problem CYCLE. Great improvement in NZFRI and DEGEN3 outweighs this bad problem. The rule executes well on the PILOT set, dramatically reducing iterations and time on three of these very difficult problems. Frequent reductions in iterations only occasionally reduce CPU time for the STAIRCASE and STANFORD sets. The SHIP set continues to stymie all of the new pivot rules, as, once again, both iteration counts and times exceed those of the usual rule. Results are mixed on the MISCELLANEOUS set. GROW22 emerges as the worst problem encountered for this rule. On the positive side, it is encouraging to note improvement on CZPROB. The ratio of iterations to rows for the normal rule on CZPROB is about 2.0, and very few degenerate pivots occur. Thus, pivot rule (3.3) enhances performance even though the standard method handled the problem quite effectively.

Figure 4-9 contains the results of pivot rule (3.3) for scaled problems. Except for the PILOT set, scaling has very little effect on relative performance. With respect to the PILOT set, the new pivot rule still outperforms the standard rule on average. The improvement is less dramatic than on unscaled problems, but it is nonetheless noteworthy given the benefits of scaling for the usual rule.

4.5. Nonlinear Estimation of the Step Length

Figure 4-10 displays results for pivot rule (3.6) on unscaled problems. (3.6) utilizes a nonlinear function to estimate step lengths. Positive results for the KETRON and PILOT sets resemble those for pivot rule (3.3). In general, (3.6) does not perform well on the STAIRCASE set; SCSD8 is particularly discouraging. It yields a

slight overall reduction in iterations for the STANFORD set, but times typically increase. Like the other pivot rules, it performs unfavorably on the SHIP set. Results vary drastically on the MISCELLANEOUS set. It performs quite well on CZPROB and NESM. The improvement for NESM is encouraging since the other new rules failed to reduce CPU time. However, it performs quite poorly on 80BAU3B, as it more than doubles the iterations and triples the time. This is the only example where one of the new rules increased the ratio of iterations to rows to above 5.0.

Figure 4-11 summarizes the performance of (3.6) on scaled problems. Only minor differences from the unscaled results occur. Even the PILOT set reveals only a moderate decline in relative efficiency.

4.6. Parametric Method

Figure 4-12 shows results for the standard parametric algorithm outlined in Figure 3-1. The algorithm performs quite well on the KETRON set. This confirms our intuition since the algorithm's purpose is to make progress in the dual even when stalled in the primal. The results for NZFRI are particularly encouraging. Notice also that the parametric algorithm requires less additional work per iteration than the previously tested rules. The algorithm exhibits tremendous success on the PILOT set, as it solves each problem at least twice as quickly. Notice that it solved PILOTJA more than eight times faster. This performance is not particularly surprising since the parametric algorithm's choice of incoming variable remains invariant under column scaling. Given the poor scaling present in these problems, one might anticipate the benefit of a unit-free pivot rule. Nonetheless, we shall see that the rule still performs well when MINOS scales these problems. Mixed results characterize the STANFORD set, as we see good performances on CAPRI and E226 accompanied by disappointing ones for BRANDY and ETAMACRO. Results for the SHIP set strongly resemble those for the other pivot rules. None of the two-objective strategies seems to work well. Performance varies drastically on the MISCELLANEOUS set. The rule does quite well on STANDATA, VTPBASE and FFFFF800. However, an alarming trend emerges for the problems GROW7, GROW15, and GROW22. The same model generates each of these problems; only the number of time periods changes. Performance worsens as size increases. The parametric method requires over six times more CPU time to solve GROW22, a figure far beyond the worst problems for any of the other pivot rules. Nonetheless, except for these three problems, the algorithm doesn't increase CPU time by more than fifty percent and works quite well on most of the larger, more difficult problems.

Figure 4-13 contains results for the standard parametric algorithm on scaled problems. Although overall relative performance declines compared to the results without scaling, the approach still does well on most of the larger problems. With respect to the KETRON set, scaling had little influence on eight of the nine problems, and relative performance remained quite favorable. However, on WOODW the parametric algorithm required much more time than the simplex method, as the ratio of CPU times exceeded five. Contrast this with the unscaled results, where the parametric algorithm solved this problem substantially faster than the simplex method. Nonetheless, considering all problems equally, the parametric algorithm performs favorably on this set. As for the PILOT set, relative performance declines substantially compared to unscaled results, but the algorithm still outperforms the simplex method on all four problems. Scaling results in only minor differences on the remaining test sets. Note that the parametric algorithm processes GROW15 and GROW22 much more effectively when scaled, although it still requires more time than the simplex method.

Since the parametric algorithm is designed particularly for degenerate problems, we again examine the frequency of degenerate pivots. Figures 4-14 and 4-15

show the results for unscaled and scaled problems. Unlike the Degeneracy Screen, it does not usually reduce the frequency of blocked pivots. Consider the problem DEGEN3. The algorithm dramatically outdoes the simplex method, yet the percentage of blocked pivots increases slightly. On DEGEN2 it outperforms the simplex method despite a significant increase in the frequency; one encounters similar results for SEBA and 80BAU3B. This characteristic motivated the modified parametric algorithm of Figure 3-2.

4.7. Summary

Summarizing the results, the Degeneracy Screen appears to be the best of the pivot rules that utilize a dynamic second objective vector. It decreases iteration counts on the vast majority of problems tested. Even when iterations increase, times almost never exceed those of the standard simplex method by a factor greater than 1.3. Also, the instances where relative performance was poorest consisted of small to moderately sized problems that the standard method solved quite efficiently. This contrasts with the new rule's ability to substantially reduce iterations and time on highly degenerate problems. Many of these are the large, difficult problems that require large amounts of CPU time when solved by the regular method. The Degeneracy Screen decreases the susceptibility of the simplex method to such disasters, and it does so at minimal risk.

The main drawback of this rule is that it is unlikely to do well on problems with few blocked pivots like CZPROB and PILOTS. Pivot rules (3.2), (3.3), and (3.6) attempt to alleviate this problem. They can succeed in solving fairly nondegenerate linear programs quickly, and they perform extremely well on the PILOT set, another group of very difficult, time consuming problems. Unfortunately, as the potential for savings improves, so does the possibility of substantial increases in time. The worst relative performances may occur on larger problems, and the CPU time may exceed 1.5 that of the standard rule. In rare cases the rules require twice as much time. Despite the risks, these rules still make the simplex method less prone to disaster since they work well on most of the large, difficult problems. Similar conclusions arise for the parametric algorithm, but the variation in performance is much greater. The algorithm exhibits the ability to improve or worsen times by factors greater than six. The risk increases, but so does the payoff.

Figure 4-1. Test Problem Sizes

(KETRON)	
Problem Name	Size
DEGEN1	67 x 72
KB2	46 x 41
WOOD1P	486 x 2594
DEGEN2	445 x 534
TUFF	371 x 587
WOODW	1099 x 8405
CYCLE	2234 x 2857
NZFRI	624 x 3521
DEGEN3	1504 x 1818

(PILOT)	
Problem Name	Size
PILOT4	411 x 1000
PILOTWE	723 x 2789
PILOTS	1442 x 3652
PILOTJA	941 x 1988

(STAIRCASE)	
Problem Name	Size
SCAGR7	130 x 140
SCORPION	389 x 358
SC205	206 x 203
SCSD1	78 x 760
SCTAP1	301 x 480
SCFXM1	331 x 457
SCAGR25	472 x 500
SCSD6	148 x 1350
SCFXM2	661 x 914
SCRS8	491 x 1169
SCSD8	398 x 2750
SCFXM3	991 x 1371
SCTAP2	1091 x 1880
SCTAP3	1481 x 2480

(STANFORD)	
Problem Name	Size
AFIRO	28 x 32
SHARE2B	97 x 79
BEACONFD	174 x 262
CAPRI	272 x 353
BRANDY	221 x 249
ADLITTLE	57 x 97
SHARE1B	118 x 225
ISRAEL	175 x 142
BANDM	306 x 472
STAIR	357 x 467
ETAMACRO	401 x 688
E226	224 x 282

(SHIP)	
Problem Name	Size
SHIP04S	403 x 1458
SHIP04L	403 x 2118
SHIP08S	779 x 2387
SHIP12S	1152 x 2763
SHIP08L	779 x 4283
SHIP12L	1152 x 5427

(MISC.)	
Problem Name	Size
RECIPE	92 x 180
BORE3D	234 x 315
GROW7	141 x 301
SEBA	516 x 1028
SHELL	537 x 1775
STANDATA	360 x 1075
VTPBASE	199 x 203
GROW15	301 x 645
GANGES	1310 x 1681
GFRDPNC	617 x 1092
GROW22	441 x 946
SIERRA	1228 x 2036
FFFFF800	525 x 854
CZPROB	930 x 3523
NESM	663 x 2923
80BAU3B	2263 x 9799
25FV47	822 x 1571

Figure 4-2. Results for (3.1) on Unscaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
DEGEN1	17	15	1.13	3.89	4.64	0.84
KB2	58	65	0.89	6.38	6.41	1.00
WOOD1P	564	564	1.00	1575.32	1382.79	1.14
DEGEN2	854	1264	0.68	435.09	517.99	0.84
TUFF	553	1407	0.39	259.20	527.04	0.49
WOODW	1544	2381	0.65	4348.36	5590.70	0.78
CYCLE	2974	3433	0.87	6627.70	6962.93	0.95
NZFRI	2808	10970	0.26	3598.46	11164.49	0.32
DEGEN3	4067	11096	0.37	7859.09	18027.67	0.44
		Geom. Mean:	0.62		Geom. Mean:	0.70

(PILOT)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
PILOT4	2622	3811	0.69	1807.78	2069.30	0.87
PILOTWE	9427	15730	0.60	11853.12	15527.70	0.76
PILOTS	38578	40241	0.96	145162.54	136832.00	1.06
PILOTJA	27697	50563	0.55	43321.84	64719.70	0.67
		Geom. Mean:	0.68		Geom. Mean:	0.83

(STAIRCASE)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
SCAGR7	89	91	0.98	15.35	13.90	1.10
SCORPION	138	138	1.00	50.23	44.40	1.13
SC205	120	139	0.86	26.11	25.80	1.01
SCSD1	218	293	0.74	67.44	74.20	0.91
SCTAP1	322	389	0.83	96.00	95.10	1.01
SCFXM1	410	393	1.04	129.87	108.30	1.20
SCAGR25	420	472	0.89	164.53	157.10	1.05
SCSD6	416	804	0.52	193.06	295.50	0.65
SCFXM2	853	848	1.01	507.38	424.50	1.20
SCRS8	862	1003	0.86	446.72	430.20	1.04
SCSD8	1169	1219	0.96	1035.76	912.20	1.14
SCFXM3	1249	1355	0.92	1109.35	1023.70	1.08
SCTAP2	1119	1503	0.74	1053.04	1198.30	0.88
SCTAP3	1679	1755	0.96	2113.72	1821.60	1.16
		Geom. Mean:	0.87		Geom. Mean:	1.03

Figure 4-2(ctd). Results for (3.1) on Unscaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
AFIRO	9	9	1.00	2.33	2.36	0.99
SHARE2B	132	105	1.26	17.37	13.55	1.28
BEACONFD	87	87	1.00	32.34	27.03	1.20
CAPRI	278	320	0.87	76.57	72.27	1.06
BRANDY	319	296	1.08	91.68	71.42	1.28
ADLITTLE	123	144	0.85	12.88	13.43	0.96
SHARE1B	309	286	1.08	53.21	40.50	1.31
ISRAEL	404	298	1.36	90.63	60.59	1.50
BANDM	319	445	0.72	117.45	134.18	0.88
STAIR	449	526	0.85	240.06	251.21	0.96
ETAMACRO	491	640	0.77	166.86	180.22	0.93
E226	639	581	1.10	180.82	137.79	1.31
		Geom. Mean:	0.98		Geom. Mean:	1.12

(SHIP)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
SHIP04S	156	148	1.05	108.13	91.30	1.18
SHIP04L	233	220	1.06	201.12	159.10	1.26
SHIP08S	252	246	1.02	283.14	231.20	1.22
SHIP12S	419	448	0.94	538.16	478.20	1.13
SHIP08L	453	449	1.01	728.93	561.40	1.30
SHIP12L	929	873	1.06	1711.54	1344.20	1.27
		Geom. Mean:	1.02		Geom. Mean:	1.23

(MISC.)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
RECIPE	33	33	1.00	8.88	8.60	1.03
BORE3D	131	111	1.18	35.36	29.94	1.18
GROW7	167	167	1.00	59.44	53.60	1.11
SEBA	175	212	0.83	104.80	106.80	0.98
SHELL	239	258	0.93	157.41	143.23	1.10
STANDATA	218	373	0.58	98.28	130.70	0.75
VTPBASE	236	423	0.56	46.14	65.40	0.71
GROW15	509	512	0.99	311.54	262.80	1.19
GANGES	684	678	1.01	611.28	477.71	1.28
GFRDPNC	699	682	1.02	336.89	278.86	1.21
GROW22	956	901	1.06	823.44	630.30	1.31
SIERRA	926	1316	0.70	942.93	1145.80	0.82
FFFFF800	657	2027	0.32	358.07	878.63	0.41
CZPROB	1720	1841	0.93	2040.57	1773.07	1.15
NESM	4849	5153	0.94	4621.95	4051.30	1.14
80BAU3B	8369	8059	1.04	23516.88	19078.97	1.23
25FV47	8816	9072	0.97	10610.17	8721.53	1.22
		Geom. Mean:	0.85		Geom. Mean:	1.01

Figure 4-3. Blocked Pivots for (3.1) on Unscaled Problems

(KETRON)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
DEGEN1	17	6	35.3	15	6	40.0
KB2	58	22	37.9	65	25	38.5
WOOD1P	564	310	55.0	564	328	58.2
DEGEN2	854	442	51.8	1264	776	61.4
TUFF	553	246	44.5	1407	554	39.4
WOODW	1544	587	38.0	2381	891	37.4
CYCLE	2974	2806	94.4	3433	3209	93.5
NZFRI	2808	733	26.1	10970	5677	51.8
DEGEN3	4067	2539	62.4	11096	9038	81.5

(PILOT)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
PILOT4	2622	117	4.5	3811	562	14.7
PILOTWE	9427	303	3.2	15730	2406	15.3
PILOTS	38578	479	1.2	40247	1192	3.0
PILOTJA	27697	1324	4.8	50563	10613	21.0

(STAIRCASE)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
SCAGR7	89	9	10.1	91	9	9.9
SCORPION	138	58	42.0	138	58	42.0
SC205	120	4	3.3	139	14	10.1
SCSD1	218	96	44.0	293	127	43.3
SCTAP1	322	50	15.5	389	106	27.2
SCFXM1	410	99	24.1	393	112	28.5
SCAGR25	420	52	12.4	472	70	14.8
SCSD6	416	148	35.6	804	311	38.7
SCFXM2	853	164	19.2	848	235	27.7
SCRS8	862	175	20.3	1003	200	19.9
SCSD8	1169	386	33.0	1219	569	46.7
SCFXM3	1249	268	21.5	1355	365	26.9
SCTAP2	1119	446	39.9	1503	613	40.8
SCTAP3	1679	693	41.3	1755	911	51.9

Figure 4-3(ctd). Blocked Pivots for (3.1) on Unscaled Problems

(STANFORD)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
AFIRO	9	5	55.6	9	5	55.6
SHARE2B	132	21	15.9	105	22	21.0
BEACONFD	87	6	6.9	87	6	6.9
CAPRI	278	14	5.0	320	25	7.8
BRANDY	319	23	7.2	296	36	12.2
ADLITTLE	123	10	8.1	144	18	12.5
SHARE1B	309	0	0.0	286	2	0.7
ISRAEL	404	29	7.2	298	52	17.4
BANDM	319	18	5.6	445	36	8.1
STAIR	449	15	3.3	526	31	5.9
ETAMACRO	491	138	28.1	640	210	32.8
E226	639	32	5.0	581	83	14.3

(SHIP)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
SHIP04S	156	34	21.8	148	28	18.9
SHIP04L	233	45	19.3	220	37	16.8
SHIP08S	252	66	26.2	246	61	24.8
SHIP12S	419	80	19.1	448	86	19.2
SHIP08L	453	78	17.2	449	68	15.1
SHIP12L	929	173	18.6	873	177	20.3

(MISC.)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
RECIPE	33	3	9.1	33	3	9.1
BORE3D	131	97	74.0	111	90	81.1
GROW7	167	3	1.8	167	7	4.2
SEBA	175	18	10.3	212	31	14.6
SHELL	239	43	18.0	258	48	18.6
STANDATA	218	119	54.6	373	247	66.2
VTPBASE	236	49	20.8	423	160	37.8
GROW15	509	7	1.4	512	22	4.3
GANGES	684	140	20.5	678	148	21.8
GFRDPNC	669	267	39.9	682	329	48.2
GROW22	956	12	1.3	901	41	4.6
SIERRA	926	460	49.7	1316	639	48.6
FFFFF800	657	141	21.5	2027	615	30.3
CZPROB	1720	30	1.7	1841	57	3.1
NESM	4849	1	0.0	5153	1	0.0
80BAU3B	8369	373	4.5	8059	791	9.8
25FV47	8816	121	1.4	9072	720	7.9

Figure 4-4. Results for (3.1) on Scaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
DEGEN1	17	15	1.13	5.16	4.97	1.04
KB2	55	55	1.00	6.46	5.98	1.08
WOOD1P	693	872	0.79	1958.89	2047.52	0.96
DEGEN2	612	1276	0.48	321.95	523.26	0.62
TUFF	482	1124	0.43	222.59	401.42	0.55
WOODW	2320	3801	0.61	5913.36	7933.99	0.75
CYCLE	2519	3017	0.83	5506.03	5912.14	0.93
NZFRI	1832	7454	0.25	2173.41	6904.42	0.31
DEGEN3	4914	10453	0.47	9236.65	17283.76	0.53
		Geom Mean:	0.61		Geom Mean:	0.70

(PILOT)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
PILOT4	1261	1533	0.82	836.61	841.00	0.99
PILOTWE	3847	6696	0.57	4746.18	6457.30	0.74
PILOTS	12923	18165	0.71	71281.88	84860.00	0.84
PILOTJA	4788	7114	0.67	6596.59	7751.90	0.85
		Geom Mean:	0.69		Geom Mean:	0.85

(STAIRCASE)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
SCAGR7	85	88	0.97	15.20	14.30	1.06
SCORPION	104	104	1.00	42.03	38.30	1.10
SC205	116	110	1.05	26.37	22.30	1.18
SCSD1	421	623	0.68	117.38	145.30	0.81
SCTAP1	219	216	1.01	69.77	61.40	1.14
SCFXM1	329	315	1.04	111.88	92.90	1.20
SCAGR25	300	307	0.98	121.28	105.40	1.15
SCSD6	761	1561	0.49	342.02	553.40	0.62
SCFXM2	750	874	0.86	467.02	441.20	1.06
SCRS8	562	668	0.84	307.25	302.30	1.02
SCSD8	2021	4335	0.47	1799.47	3288.10	0.55
SCFXM3	1135	1223	0.93	1028.57	915.20	1.12
SCTAP2	745	753	0.99	696.66	602.50	1.16
SCTAP3	948	944	1.00	1140.78	988.30	1.15
		Geom Mean:	0.85		Geom Mean:	1.00

Figure 4-4(ctd). Results for (3.1) on Scaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
AFIRO	6	6	1.00	2.43	2.60	0.93
SHARE2B	117	115	1.02	16.55	14.40	1.15
BEACONFD	97	98	0.99	36.76	33.10	1.11
CAPRI	238	245	0.97	66.09	57.30	1.15
BRANDY	361	477	0.76	101.24	111.50	0.91
ADLITTLE	110	114	0.96	12.80	12.00	1.07
SHARE1B	277	274	1.01	45.96	40.60	1.13
ISRAEL	251	296	0.85	59.21	63.30	0.94
BANDM	385	454	0.85	140.21	139.70	1.00
STAIR	465	418	1.11	295.29	253.40	1.17
ETAMACRO	386	618	0.62	147.06	190.80	0.77
E226	558	472	1.18	155.40	112.90	1.38
		Geom. Mean:	0.93		Geom. Mean:	1.05

(SHIP)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
SHIP04S	155	144	1.08	115.50	95.70	1.21
SHIP04L	249	231	1.08	214.90	171.70	1.25
SHIP08S	264	240	1.10	292.82	232.20	1.26
SHIP12S	407	399	1.02	532.92	436.00	1.22
SHIP08L	461	449	1.03	747.43	576.80	1.30
SHIP12L	911	869	1.05	1790.64	1384.40	1.29
		Geom. Mean:	1.06		Geom. Mean:	1.25

(MISC.)	ITERATIONS			CPU		
Problem	Deg. Screen	Simplex	Deg./Simp	Deg. Screen	Simplex	Deg./Simp
RECIPE	33	33	1.00	9.30	9.40	0.99
BORE3D	96	148	0.65	31.45	39.30	0.80
GROW7	151	160	0.94	56.67	53.20	1.07
SEBA	317	364	0.87	155.81	158.60	0.98
SHELL	238	258	0.92	162.03	150.20	1.08
STANDATA	135	129	1.05	71.61	63.60	1.13
VTPBASE	46	89	0.52	15.96	20.40	0.78
GROW15	457	464	0.98	274.61	235.60	1.17
GANGES	715	699	1.02	661.01	529.50	1.25
GFRDPNC	720	659	1.09	349.04	288.20	1.21
GROW22	693	756	0.92	573.65	507.70	1.13
SIERRA	891	1351	0.66	894.51	1148.90	0.78
FFFFF800	827	939	0.88	442.36	435.10	1.02
CZPROB	1295	1525	0.85	1593.86	1519.50	1.05
NESM	3359	2887	1.16	3283.11	2377.50	1.38
80BAU3B	13949	17466	0.80	38422.39	37894.10	1.01
25FV47	6757	8442	0.80	7505.33	7684.40	0.98
		Geom. Mean:	0.90		Geom. Mean:	1.04

Figure 4-5. Blocked Pivots for (3.1) on Scaled Problems

(KETRON)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
DEGEN1	17	6	35.3	15	6	40.0
KB2	55	14	25.5	55	14	25.5
WOOD1P	693	406	58.6	872	542	62.2
DEGEN2	612	252	41.2	1276	729	57.1
TUFF	482	208	43.2	1124	406	36.1
WOODW	2320	898	38.7	3801	1811	47.6
CYCLE	2519	2174	86.3	3017	2564	85.0
NZFRI	1832	642	35.0	7454	3779	50.7
DEGEN3	4914	3559	72.4	10453	8195	78.4

(PILOT)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
PILOT4	1261	53	4.2	1533	186	12.1
PILOTWE	3847	62	1.6	6696	1090	16.3
PILOTS	12923	541	4.2	18165	2044	11.3
PILOTJA	4788	177	3.7	7114	613	8.6

(STAIRCASE)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
SCAGR7	85	9	10.6	88	10	11.4
SCORPION	104	44	42.3	104	44	42.3
SC205	116	8	6.9	110	14	12.7
SCSD1	421	202	48.0	623	394	63.2
SCTAP1	219	58	26.5	216	60	27.8
SCFXM1	329	33	10.0	315	41	13.0
SCAGR25	300	46	15.3	307	48	15.6
SCSD6	761	185	24.3	1561	606	38.8
SCFXM2	750	69	9.2	874	118	13.5
SCRS8	562	90	16.0	668	180	26.9
SCSD8	2021	633	31.3	4335	2722	62.8
SCFXM3	1135	111	9.8	1223	183	15.0
SCTAP2	745	384	51.5	753	419	55.6
SCTAP3	948	566	59.7	944	564	59.7

Figure 4-5(ctd). Blocked Pivots for (3.1) on Scaled Problems

(STANFORD)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
AFIRO	6	3	50.0	6	3	50.0
SHARE2B	117	17	14.5	115	20	17.4
BEACONFD	97	12	12.4	98	17	17.3
CAPRI	238	17	7.1	245	25	10.2
BRANDY	361	11	3.0	477	28	5.9
ADLITTLE	110	9	8.2	114	13	11.4
SHARE1B	277	4	1.4	274	10	3.6
ISRAEL	251	9	3.6	296	17	5.7
BANDM	385	14	3.6	454	28	6.2
STAIR	465	19	4.1	418	52	12.4
ETAMACRO	386	66	17.1	618	122	19.7
E226	558	39	7.0	472	91	19.3

(SHIP)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
SHIP04S	155	30	19.4	144	25	17.4
SHIP04L	249	42	16.9	231	36	15.6
SHIP08S	264	80	30.3	240	60	25.0
SHIP12S	407	90	22.1	399	68	17.0
SHIP08L	461	87	18.9	449	73	16.3
SHIP12L	911	179	19.6	869	171	19.7

(MISC.)	DEG. SCREEN			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
RECIPE	33	3	9.1	33	3	9.1
BORE3D	96	41	42.7	148	85	57.4
GROW7	151	5	3.3	160	6	3.8
SEBA	317	49	15.5	364	58	15.9
SHELL	238	45	18.9	258	48	18.6
STANDATA	135	75	55.6	129	74	57.4
VTPBASE	46	9	19.6	89	38	42.7
GROW15	457	6	1.3	464	12	2.6
GANGES	715	165	23.1	699	204	29.2
GFRDPNC	720	237	32.9	659	271	41.1
GROW22	693	10	1.4	756	23	3.0
SIERRA	891	399	44.8	1351	552	40.9
FFFFF800	827	169	20.4	939	346	36.8
CZPROB	1295	43	3.3	1525	94	6.2
NESM	3359	0	0.0	2887	0	0.0
80BAU3B	13949	521	3.7	17466	1177	6.7
25FV47	6757	116	1.7	6442	714	8.5

Figure 4-6. Results for (3.2) on Unscaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
DEGEN1	17	15	1.13	4.81	4.64	1.04
KB2	64	65	0.98	7.05	6.41	1.10
WOOD1P	533	564	0.95	1532.71	1382.79	1.11
DEGEN2	722	1264	0.57	393.03	517.99	0.76
TUFF	1428	1407	1.01	673.90	527.04	1.28
WOODW	1476	2381	0.62	4265.33	5590.70	0.76
CYCLE	3317	3433	0.97	7571.84	6962.93	1.09
NZFRI	8716	10970	0.79	11215.05	11164.49	1.00
DEGEN3	5478	11096	0.49	10769.93	18027.67	0.60
		Geom. Mean:	0.80		Geom. Mean:	0.95

(PILOT)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
PILOT4	1893	3811	0.50	1334.59	2069.30	0.64
PILOTWE	10256	15730	0.65	13683.23	15527.70	0.88
PILOTS	34552	40247	0.86	137199.12	136832.00	1.00
PILOTJA	15897	50563	0.31	26552.00	64719.70	0.41
		Geom. Mean:	0.54		Geom. Mean:	0.69

(STAIRCASE)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
SCAGR7	78	91	0.86	14.37	13.90	1.03
SCORPION	121	138	0.88	47.69	44.40	1.07
SC205	146	139	1.05	34.25	25.80	1.33
SCSD1	235	293	0.80	72.21	74.20	0.97
SCTAP1	391	389	1.01	123.75	95.10	1.30
SCFXM1	479	393	1.22	170.42	108.30	1.57
SCAGR25	428	472	0.91	197.57	157.10	1.26
SCSD6	494	804	0.61	231.11	295.50	0.78
SCFXM2	1080	848	1.27	724.43	424.50	1.71
SCRS8	918	1003	0.92	578.60	430.20	1.34
SCSD8	1350	1219	1.11	1237.74	912.20	1.36
SCFXM3	1576	1355	1.16	1564.59	1023.70	1.53
SCTAP2	1922	1503	1.28	2039.18	1198.30	1.70
SCTAP3	1885	1755	1.06	2553.15	1821.60	1.40
		Geom. Mean:	0.99		Geom. Mean:	1.28

Figure 4-6(ctd). Results for (3.2) on Unscaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
AFIRO	9	9	1.00	2.36	2.36	1.00
SHARE2B	118	105	1.12	17.20	13.55	1.27
BEACONFD	88	87	1.01	34.29	27.03	1.27
CAPRI	383	320	1.20	108.01	72.27	1.49
BRANDY	308	296	1.04	99.77	71.42	1.40
ADLITTLE	96	144	0.67	11.70	13.43	0.87
SHARE1B	227	286	0.79	46.45	40.50	1.15
ISRAEL	260	298	0.87	73.69	60.59	1.22
BANDM	496	445	1.11	194.04	134.18	1.45
STAIR	503	526	0.96	323.27	251.21	1.29
ETAMACRO	623	640	0.97	235.27	180.22	1.31
E226	530	581	0.91	166.11	137.79	1.21
		Geom. Mean:	0.96		Geom. Mean:	1.23

(SHIP)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
SHIP04S	189	148	1.28	131.42	91.30	1.44
SHIP04L	289	220	1.31	237.23	159.10	1.49
SHIP08S	328	246	1.33	365.84	231.20	1.58
SHIP12S	480	448	1.07	661.72	478.20	1.38
SHIP08L	586	449	1.31	907.85	561.40	1.62
SHIP12L	942	873	1.08	1834.97	1344.20	1.37
		Geom. Mean:	1.22		Geom. Mean:	1.48

(MISC.)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
RECIPE	33	33	1.00	9.10	8.60	1.06
BORE3D	192	111	1.73	50.49	29.94	1.69
GROW7	196	167	1.17	78.81	53.60	1.47
SEBA	145	212	0.68	103.56	106.80	0.97
SHELL	267	258	1.03	182.24	143.23	1.27
STANDATA	360	373	0.97	153.09	130.70	1.17
VTPBASE	257	423	0.61	49.56	65.40	0.76
GROW15	586	512	1.14	419.16	262.80	1.59
GANGES	660	678	0.97	678.23	477.71	1.42
GFRDPNC	843	682	1.24	442.67	278.86	1.59
GROW22	1166	901	1.29	1142.85	630.30	1.81
SIERRA	1219	1316	0.93	1388.48	1145.80	1.21
FFFFF800	1445	2027	0.71	840.98	878.63	0.96
CZPROB	1439	1841	0.78	1874.40	1773.07	1.06
NESM	4619	5153	0.90	4730.09	4051.30	1.17
80BAU3B	5743	8059	0.71	18236.62	19078.97	0.96
25FV47	5554	9072	0.61	7420.03	8721.53	0.85
		Geom. Mean:	0.93		Geom. Mean:	1.20

Figure 4-7. Results for (3.2) on Scaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
DEGEN1	17	15	1.13	5.19	4.97	1.04
KB2	50	55	0.91	6.31	5.98	1.06
WOOD1P	724	872	0.83	2064.53	2047.52	1.01
DEGEN2	830	1276	0.65	435.95	523.26	0.83
TUFF	931	1124	0.83	432.37	401.42	1.08
WOODW	3306	3801	0.87	8698.37	7933.99	1.10
CYCLE	2723	3017	0.90	6382.93	5912.14	1.08
NZFRI	4110	7454	0.55	4957.61	6904.42	0.72
DEGEN3	3717	10453	0.36	7564.16	17283.76	0.44
		Geom. Mean:	0.75		Geom. Mean:	0.90

(PILOT)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
PILOT4	1821	1533	1.19	1252.20	841.00	1.49
PILOTWE	3626	6696	0.54	4801.64	6457.30	0.74
PILOTS	12632	18165	0.70	70873.46	84860.00	0.84
PILOTJA	7158	7114	1.01	10272.01	7751.90	1.33
		Geom. Mean:	0.82		Geom. Mean:	1.05

(STAIRCASE)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
SCAGR7	81	88	0.92	15.58	14.30	1.09
SCORPION	102	104	0.98	44.26	38.30	1.16
SC205	112	110	1.02	30.37	22.30	1.36
SCSD1	384	623	0.62	111.53	145.30	0.77
SCTAP1	265	216	1.23	87.12	61.40	1.42
SCFXM1	414	315	1.31	150.94	92.90	1.62
SCAGR25	412	307	1.34	191.06	105.40	1.81
SCSD6	864	1561	0.55	401.45	553.40	0.73
SCFXM2	912	874	1.04	629.58	441.20	1.43
SCRS8	473	668	0.71	281.04	302.30	0.93
SCSD8	2475	4335	0.57	2319.54	3288.10	0.71
SCFXM3	1478	1223	1.21	1492.70	915.20	1.63
SCTAP2	798	753	1.06	784.83	602.50	1.30
SCTAP3	967	944	1.02	1238.24	988.30	1.25
		Geom. Mean:	0.93		Geom. Mean:	1.18

Figure 4-7(ctd). Results for (3.2) on Scaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
ARRO	6	6	1.00	2.52	2.60	0.97
SHARE2B	92	115	0.80	15.13	14.40	1.05
BEACONFD	98	98	1.00	38.58	33.10	1.17
CAPRI	311	245	1.27	88.93	57.30	1.55
BRANDY	361	477	0.76	112.39	111.50	1.01
ADLITTLE	96	114	0.84	12.25	12.00	1.02
SHARE1B	144	274	0.53	31.09	40.60	0.77
ISRAEL	250	296	0.84	68.62	63.30	1.08
BANDM	474	454	1.04	187.74	139.70	1.34
STAIR	453	418	1.08	333.95	253.40	1.32
ETAMACRO	398	618	0.64	166.00	190.80	0.87
E226	400	472	0.85	128.15	112.90	1.14
		Geom. Mean:	0.87		Geom. Mean:	1.09

(SHIP)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
SHIP04S	184	144	1.28	131.24	95.70	1.37
SHIP04L	295	231	1.28	241.24	171.70	1.41
SHIP08S	322	240	1.34	363.67	232.20	1.57
SHIP12S	493	399	1.24	670.85	436.00	1.54
SHIP08L	546	449	1.22	863.01	576.80	1.50
SHIP12L	932	869	1.07	1905.47	1384.40	1.38
		Geom. Mean:	1.24		Geom. Mean:	1.46

(MISC.)	ITERATIONS			CPU		
Problem	Small Screen	Simplex	Small/Simp	Small Screen	Simplex	Small/Simp
RECIPE	33	33	1.00	9.85	9.40	1.05
BORE3D	123	148	0.83	39.25	39.30	1.00
GROW7	162	160	1.01	66.47	53.20	1.25
SEBA	300	364	0.82	166.74	158.60	1.05
SHELL	262	258	1.02	185.39	150.20	1.23
STANDATA	144	129	1.12	77.87	63.60	1.22
VTPBASE	70	89	0.79	21.61	20.40	1.06
GROW15	590	464	1.27	423.99	235.60	1.80
GANGES	751	699	1.07	759.85	529.50	1.44
GFRDPNC	656	659	1.00	352.16	288.20	1.22
GROW22	1138	756	1.51	1135.47	507.70	2.24
SIERRA	1106	1351	0.82	1218.45	1148.90	1.06
FFFFF800	1103	939	1.17	663.07	435.10	1.52
CZPROB	834	1525	0.55	1147.56	1519.50	0.76
NESM	2663	2887	0.92	2905.48	2377.50	1.22
80BAU3B	9858	17466	0.56	30330.45	37894.10	0.80
25FV47	4014	8442	0.48	4884.02	7684.40	0.64
		Geom. Mean:	0.90		Geom. Mean:	1.16

Figure 4-8. Results for (3.3) on Unscaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
DEGEN1	21	15	1.40	5.18	4.64	1.12
KB2	58	65	0.89	6.49	6.41	1.01
WOOD1P	527	564	0.93	1619.76	1382.79	1.17
DEGEN2	721	1264	0.57	398.36	517.99	0.77
TUFF	732	1407	0.52	362.12	527.04	0.69
WOODW	1923	2381	0.81	5665.30	5590.70	1.01
CYCLE	4343	3433	1.27	10233.87	6962.93	1.47
NZFRI	4781	10970	0.44	6374.83	11164.49	0.57
DEGEN3	3854	11096	0.35	7818.20	18027.67	0.43
		Geom. Mean:	0.72		Geom. Mean:	0.86

(PILOT)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
PILOT4	1908	3811	0.50	1343.11	2069.30	0.65
PILOTWE	4907	15730	0.31	6441.78	15527.70	0.41
PILOTS	40294	40247	1.00	174836.33	136832.00	1.28
PILOTJA	16897	50563	0.33	28580.93	64719.70	0.44
		Geom. Mean:	0.48		Geom. Mean:	0.62

(STAIRCASE)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
SCAGR7	107	91	1.18	18.18	13.90	1.31
SCORPION	132	138	0.96	50.88	44.40	1.15
SC205	112	139	0.81	27.37	25.80	1.06
SCSD1	224	293	0.76	72.11	74.20	0.97
SCTAP1	343	389	0.88	109.37	95.10	1.15
SCFXM1	408	393	1.04	143.79	108.30	1.33
SCAGR25	339	472	0.72	143.49	157.10	0.91
SCSD6	529	804	0.66	253.00	295.50	0.86
SCFXM2	788	848	0.93	532.80	424.50	1.26
SCRS8	758	1003	0.76	443.67	430.20	1.03
SCSD8	1062	1219	0.87	989.42	912.20	1.08
SCFXM3	1277	1355	0.94	1303.05	1023.70	1.27
SCTAP2	1223	1503	0.81	1244.67	1198.30	1.04
SCTAP3	1482	1755	0.84	1971.80	1821.60	1.08
		Geom. Mean:	0.86		Geom. Mean:	1.10

Figure 4-8(ctd). Results for (3.3) on Unscaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
AFIRO	11	9	1.22	2.45	2.36	1.04
SHARE2B	90	105	0.86	13.45	13.55	0.99
BEACONFD	90	87	1.03	34.27	27.03	1.27
CAPRI	234	320	0.73	70.41	72.27	0.97
BRANDY	299	296	1.01	94.13	71.42	1.32
ADLITTLE	110	144	0.76	11.52	13.43	0.86
SHARE1B	180	286	0.63	35.44	40.50	0.88
ISRAEL	220	298	0.74	59.47	60.59	0.98
BANDM	328	445	0.74	132.57	134.18	0.99
STAIR	424	526	0.81	271.07	251.21	1.08
ETAMACRO	564	640	0.88	216.12	180.22	1.20
E226	449	581	0.77	142.83	137.79	1.04
		Geom. Mean:	0.83		Geom. Mean:	1.04

(SHIP)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
SHIP04S	177	148	1.20	125.39	91.30	1.37
SHIP04L	259	220	1.18	225.06	159.10	1.41
SHIP08S	340	246	1.38	377.77	231.20	1.63
SHIP12S	460	448	1.03	643.78	478.20	1.35
SHIP08L	558	449	1.24	930.66	561.40	1.66
SHIP12L	1029	873	1.18	1900.62	1344.20	1.41
		Geom. Mean:	1.20		Geom. Mean:	1.47

(MISC.)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
RECIPE	34	33	1.03	8.73	8.60	1.02
BORE3D	121	111	1.09	35.67	29.94	1.19
GROW7	217	167	1.30	85.19	53.60	1.59
SEBA	203	212	0.96	125.35	106.80	1.17
SHELL	321	258	1.24	214.14	143.23	1.50
STANDATA	163	373	0.44	82.23	130.70	0.63
VTPBASE	236	423	0.56	47.59	65.40	0.73
GROW15	682	512	1.33	481.70	262.80	1.83
GANGES	647	678	0.95	645.41	477.71	1.35
GFRDPNC	712	682	1.04	376.00	278.86	1.35
GROW22	1364	901	1.51	1329.38	630.30	2.11
SIERRA	1147	1316	0.87	1302.89	1145.80	1.14
FFFFF800	1143	2027	0.56	660.00	878.63	0.75
CZPROB	1191	1841	0.65	1583.00	1773.07	0.89
NESM	4841	5153	0.94	4983.63	4051.30	1.23
80BAU3B	8121	8059	1.01	25890.15	19078.97	1.36
25FV47	5860	9072	0.65	7735.73	8721.53	0.89
		Geom. Mean:	0.90		Geom. Mean:	1.16

Figure 4-9. Results for (3.3) on Scaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
DEGEN1	21	15	1.40	5.45	4.97	1.10
KB2	41	55	0.75	5.73	5.98	0.96
WOOD1P	548	872	0.63	1811.45	2047.52	0.88
DEGEN2	612	1276	0.48	348.61	523.26	0.67
TUFF	627	1124	0.56	314.10	401.42	0.78
WOODW	2351	3801	0.62	6339.94	7933.99	0.80
CYCLE	2950	3017	0.98	7241.14	5912.14	1.22
NZFRI	2369	7454	0.32	2992.02	6904.42	0.43
DEGEN3	3571	10453	0.34	7805.75	17283.76	0.45
		Geom. Mean:	0.61		Geom. Mean:	0.77

(PILOT)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
PILOT4	1274	1533	0.83	929.67	841.00	1.11
PILOTWE	3058	6696	0.46	4189.90	6457.30	0.65
PILOTS	15991	18165	0.88	98436.64	84860.00	1.16
PILOTJA	4759	7114	0.67	7365.50	7751.90	0.95
		Geom. Mean:	0.69		Geom. Mean:	0.94

(STAIRCASE)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
SCAGR7	120	88	1.36	19.67	14.30	1.38
SCORPION	98	104	0.94	43.17	38.30	1.13
SC205	111	110	1.01	28.95	22.30	1.30
SCSD1	303	623	0.49	95.64	145.30	0.66
SCTAP1	278	216	1.29	91.42	61.40	1.49
SCFXM1	434	315	1.38	163.07	92.90	1.76
SCAGR25	417	307	1.36	182.97	105.40	1.74
SCSD6	670	1561	0.43	322.38	553.40	0.58
SCFXM2	863	874	0.99	603.57	441.20	1.37
SCRS8	367	668	0.55	218.31	302.30	0.72
SCSD8	2674	4335	0.62	2516.75	3288.10	0.77
SCFXM3	1366	1223	1.12	1425.30	915.20	1.56
SCTAP2	777	753	1.03	777.34	602.50	1.29
SCTAP3	994	944	1.05	1309.22	988.30	1.32
		Geom. Mean:	0.91		Geom. Mean:	1.15

Figure 4-9(ctd). Results for (3.3) on Scaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
AFIRO	6	6	1.00	2.47	2.60	0.95
SHARE2B	98	115	0.85	15.62	14.40	1.08
BEACONFD	89	98	0.91	36.24	33.10	1.09
CAPRI	251	245	1.02	74.70	57.30	1.30
BRANDY	410	477	0.86	134.50	111.50	1.21
ADLITTLE	104	114	0.91	13.25	12.00	1.10
SHARE1B	167	274	0.61	34.81	40.60	0.86
ISRAEL	228	296	0.77	62.12	63.30	0.98
BANDM	368	454	0.81	153.95	139.70	1.10
STAIR	305	418	0.73	247.30	253.40	0.98
ETAMACRO	366	618	0.59	158.73	190.80	0.83
E226	442	472	0.94	143.76	112.90	1.27
		Geom. Mean:	0.82		Geom. Mean:	1.05

(SHIP)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
SHIP04S	164	144	1.14	128.70	95.70	1.34
SHIP04L	245	231	1.06	219.14	171.70	1.28
SHIP08S	278	240	1.16	330.17	232.20	1.42
SHIP12S	436	399	1.09	596.33	436.00	1.37
SHIP08L	495	449	1.10	820.47	576.80	1.42
SHIP12L	902	869	1.04	1768.31	1384.40	1.28
		Geom. Mean:	1.10		Geom. Mean:	1.35

(MISC.)	ITERATIONS			CPU		
Problem	PLSE	Simplex	PLSE/Simp	PLSE	Simplex	PLSE/Simp
RECIPE	34	33	1.03	10.02	9.40	1.07
BORE3D	102	148	0.69	35.43	39.30	0.90
GROW7	166	160	1.04	79.06	53.20	1.49
SEBA	279	364	0.77	149.81	158.60	0.94
SHELL	309	258	1.20	215.01	150.20	1.43
STANDATA	81	129	0.63	56.39	63.60	0.89
VTPBASE	44	89	0.49	16.58	20.40	0.81
GROW15	484	464	1.04	378.58	235.60	1.61
GANGES	689	699	0.99	693.84	529.50	1.31
GFRDPNC	740	659	1.12	397.97	288.20	1.38
GROW22	783	756	1.04	803.14	507.70	1.58
SIERRA	1073	1351	0.79	1196.29	1148.90	1.04
FFFFF800	803	939	0.86	480.29	435.10	1.10
CZPROB	1007	1525	0.66	1390.28	1519.50	0.91
NESM	4798	2887	1.66	5147.84	2377.50	2.17
80BAU3B	11075	17466	0.63	34958.75	37894.10	0.92
25FV47	5488	8442	0.65	7065.85	7684.40	0.92
		Geom. Mean:	0.86		Geom. Mean:	1.16

Figure 4-10. Results for (3.6) on Unscaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
DEGEN1	26	15	1.73	5.48	4.64	1.18
KB2	57	65	0.88	6.63	6.41	1.03
WOOD1P	470	564	0.83	1478.03	1382.79	1.07
DEGEN2	936	1264	0.74	516.94	517.99	1.00
TUFF	964	1407	0.69	475.07	527.04	0.90
WOODW	1552	2381	0.65	4719.86	5590.70	0.84
CYCLE	3535	3433	1.03	9903.57	6962.93	1.42
NZFRI	3389	10970	0.31	4447.94	11164.49	0.40
DEGEN3	6332	11096	0.57	14607.84	18027.67	0.81
		Geom. Mean:	0.75		Geom. Mean:	0.92

(PILOT)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
PILOT4	1527	3811	0.40	1075.24	2069.30	0.52
PILOTWE	11468	15730	0.73	14835.59	15527.70	0.96
PILOTS	29941	40247	0.74	134207.69	136832.00	0.98
PILOTJA	23962	50563	0.47	38828.78	64719.70	0.60
		Geom. Mean:	0.58		Geom. Mean:	0.74

(STAIRCASE)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
SCAGR7	116	91	1.27	18.95	13.90	1.36
SCORPION	156	138	1.13	57.19	44.40	1.29
SC205	114	139	0.82	25.87	25.80	1.00
SCSD1	380	293	1.30	112.94	74.20	1.52
SCTAP1	337	389	0.87	107.17	95.10	1.13
SCFXM1	392	393	1.00	140.09	108.30	1.29
SCAGR25	602	472	1.28	235.73	157.10	1.50
SCSD6	1052	804	1.31	487.88	295.50	1.65
SCFXM2	1012	848	1.19	671.61	424.50	1.58
SCRS8	1052	1003	1.05	597.51	430.20	1.39
SCSD8	2390	1219	1.96	2260.83	912.20	2.48
SCFXM3	1648	1355	1.22	1610.65	1023.70	1.57
SCTAP2	2063	1503	1.37	2126.62	1198.30	1.77
SCTAP3	1954	1755	1.11	2656.66	1821.60	1.46
		Geom. Mean:	1.18		Geom. Mean:	1.47

Figure 4-10(ctd). Results for (3.6) on Unscaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
AFIRO	9	9	1.00	2.21	2.36	0.94
SHARE2B	102	105	0.97	15.29	13.55	1.13
BEACONFD	88	87	1.01	34.15	27.03	1.26
CAPRI	332	320	1.04	92.56	72.27	1.28
BRANDY	450	296	1.52	131.09	71.42	1.84
ADLITTLE	116	144	0.81	12.84	13.43	0.96
SHARE1B	188	286	0.66	37.09	40.50	0.92
ISRAEL	312	298	1.05	80.96	60.59	1.34
BANDM	343	445	0.77	134.78	134.18	1.00
STAIR	524	526	1.00	327.20	251.21	1.30
ETAMACRO	709	640	1.11	263.10	180.22	1.46
E226	616	581	1.06	193.48	137.79	1.40
		Geom. Mean:	0.98		Geom. Mean:	1.21

(SHIP)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
SHIP04S	171	148	1.16	126.37	91.30	1.38
SHIP04L	287	220	1.30	245.58	159.10	1.54
SHIP08S	328	246	1.33	375.59	231.20	1.62
SHIP12S	502	448	1.12	699.73	478.20	1.46
SHIP08L	512	449	1.14	862.99	561.40	1.54
SHIP12L	1029	873	1.18	2043.79	1344.20	1.52
		Geom. Mean:	1.20		Geom. Mean:	1.51

(MISC.)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
RECIPE	33	33	1.00	8.89	8.60	1.03
BORE3D	117	111	1.05	38.47	29.94	1.28
GROW7	240	167	1.44	95.00	53.60	1.77
SEBA	167	212	0.79	109.82	106.80	1.03
SHELL	364	258	1.41	232.12	143.23	1.62
STANDATA	314	373	0.84	141.06	130.70	1.08
VTPBASE	305	423	0.72	61.10	65.40	0.93
GROW15	799	512	1.56	530.60	262.80	2.02
GANGES	759	678	1.12	720.29	477.71	1.51
GFRDPNC	798	682	1.17	413.02	278.86	1.48
GROW22	1263	901	1.40	1232.41	630.30	1.96
SIERRA	1172	1316	0.89	1264.08	1145.80	1.10
FFFFF800	1388	2027	0.68	823.61	878.63	0.94
CZPROB	803	1841	0.44	1066.72	1773.07	0.60
NESM	3545	5153	0.69	3494.99	4051.30	0.86
80BAU3B	20366	8059	2.53	64370.01	19078.97	3.37
25FV47	11191	9072	1.23	14394.41	8721.53	1.65
		Geom. Mean:	1.03		Geom. Mean:	1.31

Figure 4-11. Results for (3.6) on Scaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
DEGEN1	21	15	1.40	5.13	4.97	1.03
KB2	53	55	0.96	6.58	5.98	1.10
WOOD1P	541	872	0.62	1743.81	2047.52	0.85
DEGEN2	952	1276	0.75	531.97	523.26	1.02
TUFF	586	1124	0.52	296.48	401.42	0.74
WOODW	2059	3801	0.54	5626.50	7933.99	0.71
CYCLE	3575	3017	1.18	9486.40	5912.14	1.60
NZFRI	3222	7454	0.43	4086.44	6904.42	0.59
DEGEN3	5762	10453	0.55	13372.70	17283.76	0.77
		Geom. Mean:	0.72		Geom. Mean:	0.90

(PILOT)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
PILOT4	1348	1533	0.88	900.66	841.00	1.07
PILOTWE	3467	6696	0.52	4491.04	6457.30	0.70
PILOTS	13276	18165	0.73	76898.98	84860.00	0.91
PILOTJA	5388	7114	0.76	7761.50	7751.90	1.00
		Geom. Mean:	0.71		Geom. Mean:	0.91

(STAIRCASE)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
SCAGR7	91	88	1.03	16.32	14.30	1.14
SCORPION	125	104	1.20	53.52	38.30	1.40
SC205	109	110	0.99	27.79	22.30	1.25
SCSD1	498	623	0.80	146.31	145.30	1.01
SCTAP1	329	216	1.52	104.24	61.40	1.70
SCFXM1	400	315	1.27	145.07	92.90	1.56
SCAGR25	382	307	1.24	160.53	105.40	1.52
SCSD6	1297	1561	0.83	605.33	553.40	1.09
SCFXM2	1030	874	1.18	874.94	441.20	1.53
SCRS8	515	668	0.77	300.58	302.30	0.99
SCSD8	4385	4335	1.01	4255.28	3288.10	1.29
SCFXM3	1588	1223	1.30	1524.68	915.20	1.67
SCTAP2	970	753	1.29	976.63	602.50	1.62
SCTAP3	1344	944	1.42	1772.52	988.30	1.79
		Geom. Mean:	1.11		Geom. Mean:	1.37

Figure 4-11(ctd). Results for (3.6) on Scaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
AFIRO	6	6	1.00	2.47	2.60	0.95
SHARE2B	93	115	0.81	14.84	14.40	1.03
BEACONFD	103	98	1.05	40.24	33.10	1.22
CAPRI	237	245	0.97	70.27	57.30	1.23
BRANDY	494	477	1.04	148.40	111.50	1.33
ADLITTLE	82	114	0.72	10.62	12.00	0.89
SHARE1B	134	274	0.49	28.46	40.60	0.70
ISRAEL	203	296	0.69	56.87	63.30	0.90
BANDM	416	454	0.92	165.99	139.70	1.19
STAIR	418	418	1.00	310.78	253.40	1.23
ETAMACRO	560	618	0.91	223.56	190.80	1.17
E226	467	472	0.99	148.97	112.90	1.32
		Geom. Mean:	0.86		Geom. Mean:	1.08

(SHIP)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
SHIP04S	176	144	1.22	135.29	95.70	1.41
SHIP04L	297	231	1.29	256.43	171.70	1.49
SHIP08S	321	240	1.34	367.52	232.20	1.58
SHIP12S	487	399	1.22	675.87	436.00	1.55
SHIP08L	530	449	1.18	882.77	576.80	1.53
SHIP12L	956	869	1.10	1934.15	1384.40	1.40
		Geom. Mean:	1.22		Geom. Mean:	1.49

(MISC.)	ITERATIONS			CPU		
Problem	NLSE	Simplex	NLSE/Simp	NLSE	Simplex	NLSE/Simp
RECIPE	33	33	1.00	9.96	9.40	1.06
BORE3D	101	148	0.68	35.79	39.30	0.91
GROW7	295	160	1.84	120.33	53.20	2.26
SEBA	302	364	0.83	160.44	158.60	1.01
SHELL	361	258	1.40	237.03	150.20	1.58
STANDATA	251	129	1.95	120.37	63.60	1.89
VTPBASE	47	89	0.53	16.99	20.40	0.83
GROW15	636	464	1.37	445.62	235.60	1.89
GANGES	831	699	1.19	777.18	529.50	1.47
GFRDPNC	788	659	1.20	410.19	288.20	1.42
GROW22	1125	756	1.49	1123.72	507.70	2.21
SIERRA	909	1351	0.67	987.50	1148.90	0.86
FFFFF800	1314	939	1.40	769.92	435.10	1.77
CZPROB	818	1525	0.54	1124.45	1519.50	0.74
NESM	3868	2887	1.34	3892.34	2377.50	1.64
80BAU3B	36416	17466	2.08	112980.62	37894.10	2.98
25FV47	13031	8442	1.54	15820.22	7684.40	2.06
		Geom. Mean:	1.14		Geom. Mean:	1.45

Figure 4-12. Results for Parametric Algorithm on Unscaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
DEGEN1	23	15	1.53	5.18	4.64	1.12
KB2	80	65	1.23	7.43	6.41	1.16
WOOD1P	745	564	1.32	1974.86	1382.79	1.43
DEGEN2	1062	1264	0.84	463.55	517.99	0.89
TUFF	524	1407	0.37	221.55	527.04	0.42
WOODW	1841	2381	0.77	4835.80	5590.70	0.86
CYCLE	2348	3433	0.68	4612.36	6962.93	0.66
NZFRI	2268	10970	0.21	2702.98	11164.49	0.24
DEGEN3	4921	11096	0.44	8114.37	18027.67	0.45
		Geom. Mean:	0.69		Geom. Mean:	0.70

(PILOT)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
PILOT4	1621	3811	0.43	984.32	2069.30	0.48
PILOTWE	3743	15730	0.24	4037.17	15527.70	0.26
PILOTS	17782	40247	0.44	61415.01	136832.00	0.45
PILOTJA	6205	50563	0.12	8023.20	64719.70	0.12
		Geom. Mean:	0.27		Geom. Mean:	0.29

(STAIRCASE)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
SCAGR7	102	91	1.12	16.02	13.90	1.15
SCORPION	126	138	0.91	46.13	44.40	1.04
SC205	142	139	1.02	26.59	25.80	1.03
SCSD1	233	293	0.80	69.56	74.20	0.94
SCTAP1	241	389	0.62	72.17	95.10	0.76
SCFXM1	396	393	1.01	117.93	108.30	1.09
SCAGR25	578	472	1.22	207.12	157.10	1.32
SCSD6	592	804	0.74	251.25	295.50	0.85
SCFXM2	868	848	1.02	474.46	424.50	1.12
SCRS8	490	1003	0.49	243.76	430.20	0.57
SCSD8	1709	1219	1.40	1373.20	912.20	1.51
SCFXM3	1389	1355	1.01	1090.65	1023.70	1.07
SCTAP2	774	1503	0.51	696.08	1198.30	0.58
SCTAP3	1071	1755	0.61	1247.02	1821.60	0.68
		Geom. Mean:	0.85		Geom. Mean:	0.94

Figure 4-12(ctd). Results for Parametric Algorithm on Unscaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
AFIRO	10	9	1.11	2.50	2.36	1.06
SHARE2B	123	105	1.17	15.73	13.55	1.16
BEACONFD	85	87	0.98	32.06	27.03	1.19
CAPRI	203	320	0.63	52.77	72.27	0.73
BRANDY	442	296	1.49	104.60	71.42	1.46
ADLITTLE	114	144	0.79	12.56	13.43	0.94
SHARE1B	266	286	0.93	43.08	40.50	1.06
ISRAEL	233	298	0.78	53.77	60.59	0.89
BANDM	477	445	1.07	147.85	134.18	1.10
STAIR	679	526	1.29	310.55	251.21	1.24
ETAMACRO	847	640	1.32	271.04	180.22	1.50
E226	409	581	0.70	107.53	137.79	0.78
		Geom. Mean:	0.99		Geom. Mean:	1.07

(SHIP)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
SHIP04S	214	148	1.45	128.91	91.30	1.41
SHIP04L	287	220	1.30	211.09	159.10	1.33
SHIP08S	293	246	1.19	290.71	231.20	1.26
SHIP12S	456	448	1.02	529.93	478.20	1.11
SHIP08L	571	449	1.27	778.16	561.40	1.39
SHIP12L	968	873	1.11	1565.12	1344.20	1.16
		Geom. Mean:	1.22		Geom. Mean:	1.27

(MISC.)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
RECIPE	33	33	1.00	8.82	8.60	1.03
BORE3D	110	111	0.99	31.52	29.94	1.05
GROW7	337	167	2.02	93.18	53.60	1.74
SEBA	149	212	0.70	88.85	106.80	0.83
SHELL	316	258	1.22	182.15	143.23	1.27
STANDATA	74	373	0.20	47.58	130.70	0.36
VTPBASE	144	423	0.34	28.09	65.40	0.43
GROW15	2831	512	5.53	1341.12	262.80	5.10
GANGES	792	678	1.17	638.12	477.71	1.34
GFRDPNC	610	682	0.89	287.16	278.86	1.03
GROW22	5677	901	6.30	3916.87	630.30	6.21
SIERRA	923	1316	0.70	899.67	1145.80	0.79
FFFFF800	604	2027	0.30	303.00	878.63	0.34
CZPROB	1764	1841	0.96	2019.38	1773.07	1.14
NESM	2924	5153	0.57	2676.99	4051.30	0.66
80BAU3B	6065	8059	0.75	17267.64	19078.97	0.91
25FV47	4386	9072	0.48	4261.61	8721.53	0.49
		Geom. Mean:	0.90		Geom. Mean:	1.01

Figure 4-13. Results for Parametric Algorithm on Scaled Problems

(KETRON)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
DEGEN1	20	15	1.33	5.27	4.97	1.06
KB2	47	55	0.85	5.82	5.98	0.97
WOOD1P	698	872	0.80	1986.67	2047.52	0.97
DEGEN2	966	1276	0.76	421.74	523.26	0.81
TUFF	700	1124	0.62	285.28	401.42	0.71
WOODW	15464	3801	4.07	40923.70	7933.99	5.16
CYCLE	1819	3017	0.60	3710.82	5912.14	0.63
NZFRI	2595	7454	0.35	2908.84	6904.42	0.42
DEGEN3	4793	10453	0.46	7933.65	17283.76	0.46
		Geom. Mean:	0.82		Geom. Mean:	0.89

(PILOT)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
PILOT4	1189	1533	0.78	697.10	841.00	0.83
PILOTWE	2218	6696	0.33	2395.27	6457.30	0.37
PILOTS	16471	18165	0.91	66456.58	84860.00	0.78
PILOTJA	6036	7114	0.85	7082.37	7751.90	0.91
		Geom. Mean:	0.67		Geom. Mean:	0.68

(STAIRCASE)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
SCAGR7	84	88	0.95	13.95	14.30	0.98
SCORPION	114	104	1.10	45.16	38.30	1.18
SC205	132	110	1.20	26.01	22.30	1.17
SCSD1	150	623	0.24	52.64	145.30	0.36
SCTAP1	263	216	1.22	76.12	61.40	1.24
SCFXM1	431	315	1.37	128.29	92.90	1.38
SCAGR25	356	307	1.16	127.78	105.40	1.21
SCSD6	662	1561	0.42	275.19	553.40	0.50
SCFXM2	932	874	1.07	505.08	441.20	1.14
SCRS8	513	668	0.77	258.30	302.30	0.85
SCSD8	3755	4335	0.87	2975.48	3288.10	0.90
SCFXM3	1528	1223	1.25	1217.21	915.20	1.33
SCTAP2	720	753	0.96	649.82	602.50	1.08
SCTAP3	810	944	0.86	965.38	988.30	0.98
		Geom. Mean:	0.89		Geom. Mean:	0.97

Figure 4-13(ctd). Results for Parametric Algorithm on Scaled Problems

(STANFORD)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
AFIRO	7	6	1.17	2.45	2.60	0.94
SHARE2B	102	115	0.89	14.78	14.40	1.03
BEACONFD	85	98	0.87	35.07	33.10	1.06
CAPRI	270	245	1.10	65.15	57.30	1.14
BRANDY	405	477	0.85	100.56	111.50	0.90
ADLITTLE	114	114	1.00	12.41	12.00	1.03
SHARE1B	182	274	0.66	32.46	40.60	0.80
ISRAEL	230	296	0.78	54.47	63.30	0.86
BANDM	550	454	1.21	164.56	139.70	1.18
STAIR	847	418	2.03	437.80	253.40	1.73
ETAMACRO	532	618	0.86	180.27	190.80	0.94
E226	410	472	0.87	107.68	112.90	0.95
		Geom. Mean:	0.98		Geom. Mean:	1.03

(SHIP)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
SHIP04S	156	144	1.08	112.12	95.70	1.17
SHIP04L	226	231	0.98	193.32	171.70	1.13
SHIP08S	263	240	1.10	272.17	232.20	1.17
SHIP12S	397	399	0.99	462.68	436.00	1.06
SHIP08L	483	449	1.08	705.67	576.80	1.22
SHIP12L	853	869	0.98	1531.11	1384.40	1.11
		Geom. Mean:	1.03		Geom. Mean:	1.14

(MISC.)	ITERATIONS			CPU		
Problem	Parametric	Simplex	Para./Simp	Parametric	Simplex	Para./Simp
RECIPE	33	33	1.00	9.51	9.40	1.01
BORE3D	119	148	0.80	34.66	39.30	0.88
GROW7	297	160	1.86	85.12	53.20	1.60
SEBA	370	364	1.02	169.57	158.60	1.07
SHELL	283	258	1.10	170.5	150.20	1.14
STANDATA	58	129	0.45	46.1	63.60	0.72
VTPBASE	44	89	0.49	15.97	20.40	0.78
GROW15	767	464	1.65	377.11	235.60	1.60
GANGES	789	699	1.13	637.15	529.50	1.20
GFRDPNC	613	659	0.93	286.5	288.20	0.99
GROW22	1232	756	1.63	835.34	507.70	1.65
SIERRA	1244	1351	0.92	1182.7	1148.90	1.03
FFFFF800	886	939	0.94	441.71	435.10	1.02
CZPROB	1581	1525	1.04	1816.34	1519.50	1.20
KESM	3525	2987	1.22	3319.47	2377.50	1.40
80BAU3B	8573	17466	0.49	22105.59	37894.10	0.58
25FV47	4792	8442	0.57	4489.17	7684.40	0.58
		Geom. Mean:	0.93		Geom. Mean:	1.04

Figure 4-14. Blocked Pivots for Parametric Algorithm on Unscaled Problems

(KETRON)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
DEGEN1	23	10	43.5	15	6	40.0
KB2	80	30	37.5	65	25	38.5
WOOD1P	745	650	87.2	564	328	58.2
DEGEN2	1062	757	71.3	1264	776	61.4
TUFF	524	172	32.8	1407	554	39.4
WOODW	1841	712	38.7	2381	891	37.4
CYCLE	2348	2124	90.5	3433	3209	93.5
NZFRI	2268	686	30.2	10970	5677	51.8
DEGEN3	4921	4092	83.2	11096	9038	81.5

(PILOT)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
PILOT4	1621	138	8.5	3811	562	14.7
PILOTWE	3743	590	15.8	15730	2406	15.3
PILOTS	17782	1281	7.2	40247	1192	3.0
PILOTJA	6205	1000	16.1	50563	10613	21.0

(STAIRCASE)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
SCAGR7	102	3	2.9	91	9	9.9
SCORPION	126	54	42.9	138	58	42.0
SC205	142	6	4.2	139	14	10.1
SCSD1	233	186	79.8	293	127	43.3
SCTAP1	241	63	26.1	389	106	27.2
SCFXM1	396	38	9.6	393	112	28.5
SCAGR25	578	58	10.0	472	70	14.8
SCSD6	592	275	46.5	804	311	38.7
SCFXM2	868	85	9.8	848	235	27.7
SCRS8	490	122	24.9	1003	200	19.9
SCSD8	1709	871	51.0	1219	569	46.7
SCFXM3	1369	162	11.8	1355	365	26.9
SCTAP2	774	456	58.9	1503	613	40.8
SCTAP3	1071	803	75.0	1755	911	51.9

Figure 4-14(ctd). Blocked Pivots for Parametric Algorithm on Unscaled Problems

(STANFORD)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
AFIRO	10	5	50.0	9	5	55.6
SHARE2B	123	35	28.5	105	22	21.0
BEACONFD	85	2	2.4	87	6	6.9
CAPRI	203	15	7.4	320	25	7.8
BRANDY	442	36	8.1	296	36	12.2
ADLITTLE	114	3	2.6	144	18	12.5
SHARE1B	266	2	0.8	286	2	0.7
ISRAEL	233	0	0.0	298	52	17.4
BANDM	477	50	10.5	445	36	8.1
STAIR	679	21	3.1	526	31	5.9
ETAMACRO	847	211	24.9	640	210	32.8
E226	409	54	13.2	581	83	14.3

(SHIP)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
SHIP04S	214	26	12.1	148	28	18.9
SHIP04L	287	20	7.0	220	37	16.8
SHIP08S	293	50	17.1	246	61	24.8
SHIP12S	456	69	15.1	448	86	19.2
SHIP08L	571	67	11.7	449	68	15.1
SHIP12L	968	164	16.9	873	177	20.3

(MISC.)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
RECIPE	33	3	9.1	33	3	9.1
BORE3D	110	79	71.8	111	90	81.1
GROW7	337	6	1.8	167	7	4.2
SEBA	149	32	21.5	212	31	14.6
SHELL	316	54	17.1	258	48	18.6
STANDATA	74	40	54.1	373	247	66.2
VTPBASE	144	57	39.6	423	160	37.8
GROW15	2831	46	1.6	512	22	4.3
GANGES	792	147	18.6	678	148	21.8
GFRDPNC	610	244	40.0	682	329	48.2
GROW22	5677	88	1.6	901	41	4.6
SIERRA	923	176	19.1	1316	639	48.6
FFFFF800	604	126	20.9	2027	615	30.3
CZPROB	1764	53	3.0	1841	57	3.1
NESM	2924	3	0.1	5153	1	0.0
80BAU3B	6065	938	15.5	8059	791	9.8
25FV47	4386	445	10.1	9072	720	7.9

Figure 4-15. Blocked Pivots for Parametric Algorithm on Scaled Problems

(KETRON)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
DEGEN1	20	9	45.0	15	6	40.0
KB2	47	13	27.7	55	14	25.5
WOOD1P	698	475	68.1	872	542	62.2
DEGEN2	966	599	62.0	1276	729	57.1
TUFF	700	320	45.7	1124	406	36.1
WOODW	15464	13938	90.1	3801	1811	47.6
CYCLE	1819	1641	90.2	3017	2564	85.0
NZFRI	2595	1132	43.6	7454	3779	50.7
DEGEN3	4793	3765	78.6	10453	8195	78.4

(PILOT)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
PILOT4	1189	154	13.0	1533	186	12.1
PILOTWE	2218	223	10.1	6696	1090	16.3
PILOTS	16471	1484	9.0	18165	2044	11.3
PILOTJA	6036	722	12.0	7114	613	8.6

(STAIRCASE)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
SCAGR7	84	2	2.4	88	10	11.4
SCORPION	114	42	36.8	104	44	42.3
SC205	132	7	5.3	110	14	12.7
SCSD1	150	115	76.7	623	394	63.2
SCTAP1	263	85	32.3	216	60	27.8
SCFXM1	431	48	11.1	315	41	13.0
SCAGR25	356	48	13.5	307	48	15.6
SCSD6	662	364	55.0	1561	606	38.8
SCFXM2	932	105	11.3	874	118	13.5
SCRS8	513	193	37.6	668	180	26.9
SCSD8	3755	2582	68.8	4335	2722	62.8
SCFXM3	1528	198	13.0	1223	183	15.0
SCTAP2	720	458	63.6	753	419	55.6
SCTAP3	810	555	68.5	944	564	59.7

Figure 4-15(ctd). Blocked Pivots for Parametric Algorithm on Scaled Problems

(STANFORD)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
AFIRO	7	3	42.9	6	3	50.0
SHARE2B	102	25	24.5	115	20	17.4
BEACONFD	85	10	11.8	98	17	17.3
CAPRI	270	24	8.9	245	25	10.2
BRANDY	405	37	9.1	477	28	5.9
ADLITTLE	114	2	1.8	114	13	11.4
SHARE1B	182	8	4.4	274	10	3.6
ISRAEL	230	2	0.9	296	17	5.7
BANDM	550	28	5.1	454	28	6.2
STAIR	847	56	6.6	418	52	12.4
ETAMACRO	532	87	16.4	618	122	19.7
E226	410	77	18.8	472	91	19.3

(SHIP)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
SHIP04S	156	25	16.0	144	25	17.4
SHIP04L	226	37	16.4	231	36	15.6
SHIP08S	263	53	20.2	240	60	25.0
SHIP12S	397	77	19.4	399	68	17.0
SHIP08L	483	64	13.3	449	73	16.3
SHIP12L	853	182	21.3	869	171	19.7

(MISC.)	PARAMETRIC			SIMPLEX		
Problem	Iterations	# Blocked	% Blocked	Iterations	# Blocked	% Blocked
RECIPE	33	3	9.1	33	3	9.1
BORE3D	119	54	45.4	148	85	57.4
GROW7	297	9	3.0	160	6	3.8
SEBA	370	48	13.0	364	58	15.9
SHELL	283	55	19.4	258	48	18.6
STANDATA	58	20	34.5	129	74	57.4
VTPBASE	44	10	22.7	89	38	42.7
GROW15	767	26	3.4	464	12	2.6
GANGES	789	216	27.4	699	204	29.2
GFRDPNC	613	223	36.4	659	271	41.1
GROW22	1232	70	5.7	756	23	3.0
SIERRA	1244	276	22.2	1351	552	40.9
FFFFF800	886	289	32.6	939	346	36.8
CZPROB	1581	74	4.7	1525	94	6.2
NESM	3525	6	0.2	2887	0	0.0
80BAU3B	8573	1087	12.7	17466	1177	6.7
25FV47	4792	505	10.5	8442	714	8.5

CHAPTER 5: EXTENSIONS AND FUTURE RESEARCH

This chapter briefly considers some untested ideas motivated by Chapters 2 and 3. Potential for improvement in feasible direction methods, dynamic pricing, and parametric variants of the simplex method still remains. None of the ideas is developed in detail, but the mathematics of the previous chapters should provide a foundation for future work.

5.1. Feasible Direction Methods

Much flexibility remains for the algorithms of Chapter 2. In particular, there exist criteria for selecting promising variables other than (2.13) or (2.30). However, the results of Chapter 2 suggest that any good criterion must account for the sparsity and degeneracy present in practical problems. One need not use two objectives to handle degeneracy explicitly. For example, one could apply the approach of Chang and Murty's Gravitational Method (see [4]). Consider the linear program

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \geq b. \end{aligned}$$

Assume a feasible solution x, and define

$$J(x) = \{i : A_i.x = b_i\}.$$

$J(x)$ indexes the tight constraints corresponding to x. If $J(x) = \emptyset$, x lies in the strict interior of the feasible region, so $-c$ provides a descent direction that permits a positive step length. Otherwise, use the tight constraints to formulate the following direction-finding problem:

$$\begin{aligned} \text{minimize} \quad & c^T y \\ \text{subject to} \quad & A_{J(x)}.y \geq 0, \\ & 1 - y^T y \geq 0. \end{aligned}$$

The nonlinear constraint ensures boundedness of the problem's feasible region. Its optimal solution provides a descent direction with a positive step length. The Gravitational Method does not maintain basic and nonbasic variables, but one could modify the direction-finding problem and apply it to the algorithms of Chapter 2. Specifically, consider the canonical form (1.2) and a (not necessarily basic) feasible solution x. Let

$$J(x_B) = \{i : x_{j_i} = 0\}.$$

The direction-finding problem becomes

$$\begin{aligned} \text{minimize} \quad & \overline{c}_N^T y \\ \text{subject to} \quad & \overline{A}_{J(x_B).N} y \leq 0, \\ & 1 - y^T y \geq 0. \end{aligned}$$

The optimal solution of this problem yields a descent direction with positive step length. This is extremely important given the susceptibility of feasible direction methods to degeneracy. One also sees that the approach generalizes to positive basic variables with a suitable redefinition of $J(x_B)$. The additional work required to solve the direction-finding problem may inhibit this approach. Nonetheless, it illustrates another way to alleviate problems due to degeneracy.

All of the computational tests involving feasible direction methods involve only a single column exchange in the basis during each iteration. Although this tactic guarantees a descent direction, it may result in some basic variables having substantially smaller values than nonbasic ones. Remember that the one-to-one correspondence between vertices and bases has vanished. One can associate any basis with any feasible solution, but the benefit of the descent directions generated by each basis may vary dramatically. Therefore, one wishes to associate the feasible iterate with a "good" choice of basis. Single column exchanges limit the choice of basis, so the development of computationally efficient techniques to exchange multiple columns during a single iteration emerges as an important topic. The Box Method of Cottle and Zikan (see [54]) provides insight into this problem.

5.2. Dynamic Pricing

The dynamic pricing criteria of Chapter 3 all display the ability to reduce the iterations required by the simplex method to solve practical linear programs. However, we encounter many cases where the reduction in iterations fails to reduce computation time. The need to develop additional techniques to reduce the extra work becomes apparent. Theorem 4 actually takes advantage of a degenerate pivot to reduce the computational effort in a multiple-priority pivot rule. Refer to [38] and [39] for other ways to exploit degeneracy in the simplex method. Mathematical results such as Theorem 4 should help, but one must also realize that MINOS was developed for the single-objective format of the standard simplex method. The implementations tested here conformed to that design. Additional modifications in the structure of MINOS may improve the performance of multiple-objective pivot rules.

Another approach to reducing computational time involves techniques to approximate the second objective reduced costs $\overline{d}_j$. All of the implementations compute the quantities σ and $\overline{d}_j$ explicitly during each iteration. Exploration of cheaper techniques to approximate these quantities may lighten the computational load.

The encouraging performances of these new rules on difficult practical problems does not necessarily suggest good worst-case behavior. None of the pathological problems constructed (see [3], [17], [19], [24], [25] and [53]) to demonstrate worst-case behavior of established pivot rules for the simplex method relies on degeneracy. Furthermore, procedures such as (3.2), (3.3) and (3.6) attempt to emulate inexpensively the maximum-improvement pivot rule (3.7). Jeroslow [19] established exponential worst-case behavior of that rule. It therefore seems unlikely that research into this area for the pivot rules of Chapter 3 will provide any significant results.

Finally, application of a second objective to certain nonlinear programming algorithms also merits investigation. Dynamic pricing criteria significantly improved the performance of the feasible direction methods of Chapter 2. This suggests the promise of using dynamic pricing when solving nonlinear programs by the reduced-gradient method. Lemke's algorithm may also benefit from this technique. Lemke's algorithm is a pivotal algorithm that can optimize certain quadratic programs in addition to linear programs (see [41]). It solves the following linear complementarity

problem:

$$\begin{aligned} Ix - My &= q, \\ x_i y_i &= 0, \quad i = 1, \ldots, m \\ x, y &\geq 0. \end{aligned} \tag{5.1}$$

Assuming nondegeneracy, the determination of a covering vector to initiate the algorithm uniquely determines the sequence of pivot steps. However, in practice some flexibility in the pivoting procedure may arise in the presence of degeneracy. Theoretically one must appeal to the perturbation techniques used in the simplex method to resolve such ambiguities. In practice it may be possible to break ties arbitrarily without encountering a cycle; this is almost always true with respect to the simplex method. If it also holds for Lemke's algorithm, perhaps one can exploit the flexibility arising from degeneracy. In certain cases, Lemke's algorithm may not find a solution to the linear complementarity problem even though one exists; it can terminate on a ray. Freedom in selection of the entering variable permits development of pricing criteria designed to avoid such an occurrence. In particular, let $\overline{M}_{.j}$ represent a potential incoming column relative to the current basis. Termination on a ray occurs when $\overline{M}_{.j} \leq 0$. Therefore, setting $d_B = 1$, $d_N = 0$ and computing $\overline{d}_j = d_B^T \overline{M}_{.j} = \sigma^T M_{.j}$ provides a measure of the likelihood of termination on a ray if a particular variable enters the basis. When freedom to select the entering variable arises, one could utilize $\overline{d}_j$ to choose the variable least likely to result in termination on a ray.

5.3. Parametric Algorithms

The parametric algorithm of Figure 3-1 specifies an initial parametric objective vector. This is merely one of many possible legitimate initial vectors. Research into this area may reveal new initialization procedures that further enhance the performance of the parametric algorithm.

Parametric variants of the simplex method strongly resemble Lemke's algorithm applied to linear programs. In particular (see [28]), solving a linear program by Lemke's algorithm is equivalent to solving it by Dantzig's self-dual parametric algorithm. The selection of the parametric objective corresponds to the initialization of the covering vector. Given this relation, research into the proper selection of the parametric objective vector should provide insight about good initial covering vectors for Lemke's algorithm. Furthermore, the potential for the modified parametric algorithm of Figure 3-2 to avoid degenerate pivots suggests an analogous variant of Lemke's algorithm in which the covering vector changes during the course of the algorithm in a way designed to decrease the number of iterations.

CHAPTER 6: SUMMARY AND CONCLUSIONS

What conclusions emerge from this work? We have seen that sparsity and degeneracy can inhibit the progress of the reduced-gradient variants described in Chapter 2. Utilization of a dynamic second objective function helps avoid these obstacles and significantly improves the performance of such algorithms. Although the modified algorithm tested here slightly outperformed the simplex method with respect to iterations, it failed to compete in terms of computation time. The notion of using a reduced-gradient approach to solve linear programs has existed for many years, but very few computational tests have appeared in the literature. Chapter 2 provides new insight into this topic by identifying essential problems with the method, describing techniques to elude these difficulties, and establishing some computational results. Although still not competitive with the simplex method, the results achieved here substantially improve upon the status quo. Additional improvements in this type of algorithm may remain, but they must account for the sparsity and degeneracy present in practical problems.

The most important aspect of Chapter 2 is the motivation of the use of two objectives to gain useful information about nonbasic variables. Pricing out to avoid degenerate pivots has become a computationally feasible procedure. The idea generalizes to allow pricing out to avoid small pivot steps. Chapter 2 shows that these concepts enhance the performance of variants of the reduced-gradient method.

Chapter 3 addresses applications to the simplex method. Additional extensions of the two-objective approach focus on estimating the step length associated with a nonbasic variable. Explicit computation of the exact step length for each potential incoming nonbasic variable is prohibitively expensive. Dynamic pricing provides one inexpensive way to estimate such step lengths.

The computational tests in Chapter 4 show that any of the pivot rules using dynamic pricing outperform the standard simplex method with regard to iterations. Given the increasing emphasis on parallel computing, this result is useful by itself. The results are less clear with respect to computation time, as reductions in iterations frequently do not outweigh the increased work per iteration. The Degeneracy Screen (pivot rule (3.1)) remains a safe rule, as CPU times stay close to those of the standard simplex method for even the worst problems. Furthermore, it consistently performs well on highly degenerate problems. Certain applications tend to generate highly degenerate problems (for example, see [11]). Hence, one can probably anticipate cases where (3.1) will always work well. The other pivot rules display the capacity to perform well on a wider variety of problems, but this greater potential is accompanied by increased risk of poor performance. Nonetheless, all of the dynamic pivot rules perform well on the majority of the large, difficult problems that consume the most computer time.

The final conclusion from this thesis is that despite 40 years of intensive research, potential for improvement remains in the simplex method as well as other linear programming algorithms. Many researchers tend to assume that basic research in linear programming has been exhausted. The results here suggest that we do not yet fully understand the simplex method, let alone other linear programming algorithms. Uncharted territory remains to be explored.

CHAPTER 7: APPENDIX

This chapter elaborates on the specifics of the computational tests of Chapters 2 and 4. Certain aspects of the tests involve generalizations of the main results of this thesis. Inclusion of these details earlier would only have obscured the important ideas. One need not read this chapter if only interested in the major ideas; no new mathematics will appear here. However, any reader interested in bridging the gap between theory and practice present in many implementations may find this appendix useful.

Section 1 develops a procedure to determine an optimal vertex from an optimal solution. There exist several different ways to find such a vertex, including a procedure in MINOS based on the simplex method. The method defined below strongly resembles the feasible direction methods of Chapter 2. It enables those algorithms to produce an optimal vertex instead of just an optimal solution. Section 2 then extends the theory of Chapters 2 and 3 to handle the bounded variable logic of MINOS. None of the changes involves any significant additional mathematics. Nonetheless, they provide useful insight into important aspects of a large-scale linear programming code.

7.1. A Procedure to Find an Optimal Vertex From an Optimal Solution

The algorithms of Chapter 2 find an optimal solution to the linear program (1.1). However, given the interest in sensitivity analysis, an optimal vertex becomes desirable. The following procedure determines such a vertex.

Irrespective of the particular choice of promising variables, the feasible direction method terminates with an optimal solution x^* when the set P, defined in (2.13), is empty. Let B and N index the nonbasic variables associated with x^* at termination. Let

$$I = \{j \in N : \bar{c}_j = 0,\ x_j > 0\}.$$

Since $P = \emptyset$, I indexes the positive nonbasic variables of x^*. If $I = \emptyset$, x^* is the optimal vertex corresponding to B. If $I \neq \emptyset$, x^* is not a vertex. In order to identify an optimal basis, decrease the variables in I simultaneously down to zero. Let e represent a column vector of ones. In the framework of Chapter 2, set $q_P = q_I = -e$, and perform an iteration of the reduced-gradient method. The resulting solution remains optimal since $\bar{c}_j = 0$ for $j \in I$. During this process individual variables in I either attain zero or drive a basic variable to zero. In the latter case a pivot occurs. Each iteration decreases $|I|$ by at least 1, so the procedure terminates with an optimal vertex in at most $|I|$ iterations.

We now consider an iteration in detail. We wish to decrease each variable in I without violating the equality constraints $Ax = b$. In order to do so, define $q \in R^n$ by

$$\begin{aligned} q_I &= -e \\ q_{N \setminus I} &= 0 \\ Bq_B &= -A_{\cdot I} q_I. \end{aligned}$$

Update x^* by the relation

$$x^* \leftarrow x^* + \theta q. \tag{7.1}$$

How large can θ be? We wish to make θ as large as possible while maintaining nonnegativity of x_B^* and x_I^*. In other words,

$$
\begin{aligned}
& x^* + \theta q \geq 0 \\
\Rightarrow\ & \theta q_j \geq -x_j^* \qquad j = 1, \ldots, n \\
\Rightarrow\ & \theta \leq -\frac{x_j^*}{q_j} \qquad j \in B \cup I : q_j < 0 \\
\Rightarrow\ & \theta = \min_{j \in B \cup I : q_j < 0} -\frac{x_j^*}{q_j}.
\end{aligned} \tag{7.2}
$$

Let

$$
k = \operatorname*{argmin}_{j \in B \cup I : q_j < 0} -\frac{x_j^*}{q_j}.
$$

If $k \in I$, a variable in I hits zero without driving a basic variable below zero. A pivot is unnecessary. Update x^* by (7.1) and proceed with another iteration. Note that $x_k^* = 0$, so $|I|$ has decreased by at least 1. Now, suppose $k \in B$. Then a basic variable hits zero before any variable in I does. Let r index the component of B containing k. Define

$$
\hat{I} = \{j \in I : \overline{A}_{r,j} < 0\}.
$$

$\hat{I}$ indexes variables in I that can drive the r^{th} basic variable to zero. Since $k \in B$ and

$$
\theta \leq \min_{j \in I : q_j < 0} -\frac{x_j^*}{q_j} < +\infty,
$$

the ratio test (7.2) implies that $\hat{I} \neq \emptyset$. One can then choose

$$
s = \operatorname*{argmax}_{j \in \hat{I}} x_j
$$

to index the incoming variable. Then $\overline{A}_{r,s} < 0$ and $s \in \hat{I}$. The resulting pivot maintains nonsingularity of the basis. Update x^* by (7.1). Since $s \in I$ enters the basis index and $x_{j_r}^* = 0$, $|I|$ decreases by at least one. If $|I| = 0$, terminate with an optimal vertex. Otherwise, start the next iteration.

Since this procedure decreases $|I|$ by at least 1 during each iteration, it will drive all positive nonbasic variables to zero within $|I|$ iterations. Some flexibility exists within the pivoting strategies. In practice, however, the procedure was unnecessary for the majority of the problems tested, and it never required more than three pivots before termination. As a result, extensive experimentation was never pursued. Note that the computation times of Figure 2-3 include any time required by this procedure.

7.2. Extensions to Bounded Variables

For expository purposes we have considered linear programs of the form (1.1). Although one can express any linear program in this form, the transformations required to do so frequently involve extra variables and constraints, which would result

in computational inefficiencies. In practice one wishes to process linear programs in their most general form:

$$\begin{aligned} \text{minimize} \quad & c^T y \\ \text{subject to} \quad & A_E y = b_E \\ & A_G y \geq b_G \\ & A_L y \leq b_L \\ & \hat{l} \leq y \leq \hat{u}. \end{aligned} \tag{7.3}$$

Any linear program fits (7.3) without any transformations. Define $\hat{s} = (\hat{s}_E, \hat{s}_G, \hat{s}_L)$ as slacks on the constraints of (7.3). Rewrite (7.3) as

$$\begin{aligned} \text{minimize} \quad & c^T y \\ \text{subject to} \quad & A_E y + \hat{s}_E - b_E = 0 \\ & A_G y - (b_G + \hat{s}_G) = 0 \\ & A_L y + \hat{s}_L - b_L = 0 \\ & \hat{l} \leq y \leq \hat{u},\ \hat{s}_E = 0,\ \hat{s}_G, \hat{s}_L \geq 0. \end{aligned} \tag{7.4}$$

Let

$$\begin{aligned} s_E &= \hat{s}_E - b_E \\ s_G &= -(b_G + \hat{s}_G) \\ s_L &= \hat{s}_L - b_L. \end{aligned}$$

Then (7.4) is equivalent to

$$\begin{aligned} \text{minimize} \quad & c^T y \\ \text{subject to} \quad & A_E y + s_E = 0 \\ & A_G y + s_G = 0 \\ & A_L y + s_L = 0 \\ & \hat{l} \leq y \leq \hat{u}, \\ & -b_E \leq s_E \leq -b_E, \\ & -\infty \leq s_G \leq -b_G, \\ & -b_L \leq s_L \leq +\infty. \end{aligned}$$

Letting

$$A = \begin{pmatrix} A_E & I & & \\ A_G & & I & \\ A_L & & & I \end{pmatrix},$$

$$\begin{aligned} x &= (y, s_E, s_G, s_L), \\ l &= (\hat{l}, -b_E, -\infty, -b_L), \\ u &= (\hat{u}, -b_E, -b_G, +\infty), \end{aligned}$$

(7.4) becomes

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = 0 \\ & l \leq x \leq u. \end{aligned} \tag{7.5}$$

MINOS processes the general linear program (7.3) into the form (7.5). Only the bounds on the variables distinguish (7.5) from the standard form (1.1). Therefore, the remainder of the appendix focuses on generalizing the results of Chapters 2 and 3 to handle bounded variables.

Bounded variables generate some minor modifications to the feasible direction methods of Chapter 2. First of all, consider a search direction q and an updated solution $\overline{x} = x + \theta q$ as in (2.2). Equation (2.4) ensures that variables in P do not violate the nonnegativity constraints of the standard form linear program (1.1). We must now guarantee that such variables do not violate the bound constraints of (7.5). In other words, for $j \in P$,

$$\begin{aligned} & l_j \leq x_j + \theta q_j \leq u_j \\ \Leftrightarrow\ & \theta q_j \leq u_j - x_j \quad \text{and} \\ & \theta q_j \geq l_j - x_j \\ \Leftrightarrow\ & \theta \leq \frac{u_j - x_j}{q_j} \text{ for } j : q_j > 0 \quad \text{and} \\ & \theta \leq \frac{l_j - x_j}{q_j} \text{ for } j : q_j < 0 \\ \Leftrightarrow\ & \theta \leq \min \left\{ \min_{j \in P: q_j < 0} \frac{l_j - x_j}{q_j}, \min_{j \in P: q_j > 0} \frac{u_j - x_j}{q_j} \right\}. \end{aligned}$$

Similar logic applies to the basic variables in ratio test (2.5), the two-column linear program (2.12), and the pivoting tactics outlined by (2.15–2.20).

Bounded variables also influence the choice of promising variables. (2.13) now becomes

$$j \in P \text{ if } \begin{cases} \overline{c}_j < 0 \text{ and } x_j < u_j, & \text{or} \\ \overline{c}_j > 0 \text{ and } x_j > l_j. \end{cases}$$

The modified promising criterion (2.30) and pivot rules (3.1), (3.2), (3.3) and (3.6) all utilize reduced costs computed from a second objective vector d. Bounded variables affect the definition of d. Recall the choice of d_B defined by Lemma 3. For $i = 1, \ldots m$, set

$$d_{j_i} = \begin{cases} 1 & \text{if } x_{j_i} = 0, \\ 0 & \text{otherwise.} \end{cases}$$

This particular choice of d identifies potential incoming variables that would cause a degenerate pivot. In the presence of bounded variables, degeneracy occurs when a basic variable resides at one of its bounds. Also, instead of increasing from zero, a potential incoming variable x_j may increase from its lower bound l_j or decrease from its upper bound u_j. Let us examine these two cases separately. First, suppose $\overline{c}_j < 0$, so x_j increases from l_j. A degenerate pivot occurs if

$$\begin{aligned} \exists\, i : \overline{A}_{i,j} > 0,\ & x_{j_i} = l_{j_i} \text{ or} \\ \overline{A}_{i,j} < 0,\ & x_{j_i} = u_{j_i}. \end{aligned} \tag{7.6}$$

Therefore, for $i = 1, \ldots m$, set

$$d_{j_i} = \begin{cases} 1 & \text{if } x_{j_i} = l_{j_i}, \\ -1 & \text{if } x_{j_i} = u_{j_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{7.7}$$

If $\overline{d}_j = d_B^T \overline{A}_{\cdot j} = \sigma^T A_{\cdot j} > 0$, exclude x_j from consideration in order to avoid a degenerate pivot. If $\overline{c}_j > 0$, a degenerate pivot occurs if

$$\begin{aligned} \exists\, i : \overline{A}_{i,j} < 0,\ x_{j_i} = l_{j_i} \text{ or} \\ \overline{A}_{i,j} > 0,\ x_{j_i} = u_{j_i}. \end{aligned} \tag{7.8}$$

(7.8) is the opposite of (7.6), so set d_B as in (7.7), but exclude x_j from consideration if $\overline{d}_j < 0$. Thus, the extension to bounded variables still only requires a single solve of the form (2.26) for the vector σ.

Equation (7.7) describes how to determine d_B in the presence of degenerate basic variables. This suffices for pivot rule (3.1), but the criteria (2.30), (3.2), (3.3) and (3.6) consider positive basic variables as well. In such cases the value of a basic variable x_{j_i} no longer determines the value of d_{j_i}. Instead, consider the distance z_i of x_{j_i} to its closest bound:

$$z_i = \min \{x_{j_i} - l_{j_i}, u_{j_i} - x_{j_i}\} \qquad i = 1, \ldots m.$$

The values of z_i determine the values of d_{j_i}. So, for example, assuming that $z_i > 0$, one utilizes the following choice of d to implement pivot rule (3.6):

$$d_{j_i} = \begin{cases} 1/z_i & \text{if } x_{j_i} - l_{j_i} \leq u_{j_i} - x_{j_i} \\ -1/z_i & \text{if } x_{j_i} - l_{j_i} > u_{j_i} - x_{j_i} \end{cases} \qquad i = 1, \ldots m. \tag{7.9}$$

In practice, set z_i to $\epsilon > 0$ if $z_i < \epsilon$, where ϵ represents an appropriately defined tolerance. Note that if $\overline{c}_j > 0$, x_j decreases if it enters the basis. In that case one modifies pivot rule (3.6) so that nonnegative values of $\overline{d}_j$, instead of nonpositive ones, distinguish nonbasic variables with top priority. The same approach applies for pivot rules (3.2) and (3.3).

Theorem 4 illustrates an updating procedure for σ provided a degenerate pivot occurred during the previous iteration of the simplex method. Since the bounded variable form (7.5) redefines the notion of degeneracy, the value of ρ in the update (3.22) depends on the specifics of the pivot. In addition, MINOS 5.1 permits nonbasic slack variables with negative lower bounds and positive upper bounds to reside at any value between the bounds. This tactic attempts to improve the stability of the feasible iterates. It also aids recovery from singular bases and helps with restarts. This feature generates two additional types of degenerate pivot. A total of six possible types of degenerate pivot can occur. Each one is listed below, along with the corresponding value of ρ. Remember that, regardless of the specific case, a degenerate pivot during the k^{th} iteration implies that

$$d_{B_{k+1}} = d_{B_k} + \rho e_r.$$

Let s and j_r index the incoming and outgoing variables during the degenerate pivot of iteration k.

Case 1. Suppose $x_s = l_s$, and $x_{j_r} = l_{j_r}$. Both the incoming and outgoing variables reside at their lower bounds. Therefore, $d_{B_{k+1}(r)} = d_{B_k(r)}$, so $\rho = 0$.

Case 2. Suppose $x_s = l_s$, and $x_{j_r} = u_{j_r}$. The entering variable resides at its lower bound, while the outgoing variable resides at its upper bound. In this case (7.7) or (7.9) implies that $d_{B_{k+1}(r)} = -d_{B_k(r)}$, so $\rho = -2d_{B_k(r)}$.

Case 3. Suppose $x_s = u_s$, and $x_{j_r} = u_{j_r}$. This is analogous to Case 1, and $\rho = 0$.

Case 4. Suppose $x_s = u_s$, and $x_{j_r} = l_{j_r}$. This is analogous to Case 2, and $\rho = -2d_{B_k(r)}$.

Case 5. Suppose $l_s < x_s < u_s$, and $x_{j_r} = l_{j_r}$. This case can arise when the incoming variable is a slack variable initialized between its bounds. Let $\gamma = f(\min\{x_s - l_s, u_s - x_s\})$, where f is the function that determines the basic components of d (recall (2.27)). Then, $d_{B_{k+1}(r)} = \gamma = d_{B_k(r)} + \rho$, so $\rho = \gamma - d_{B_k(r)}$.

Case 6. Suppose $l_s < x_s < u_s$, and $x_{j_r} = u_{j_r}$. This is analogous to Case 5, so $\rho = \gamma - d_{B_k(r)}$.

The parametric algorithm also uses a second objective vector d, albeit a constant one. Although the generalization of Theorem 4 becomes unnecessary, bounded variables affect the pivoting procedure. Recall that the algorithm sets $d_j = \|A_{\cdot j}\|_2$ for $j \in N_0$ in order to ensure optimality with respect to the parametric objective function $\overline{c}_{N_0}(\theta) = \overline{c}_{N_0} + \theta d_{N_0} > 0$. Remember that the bounded variable form (7.5) permits nonbasic variables to reside at their upper bounds. As a result, if a nonbasic variable x_j has positive reduced cost and equals its upper bound, use $-\overline{d}_j$ instead of $\overline{d}_j$ in the denominator of the ratio test (3.10).

This concludes the appendix. Since the author did not wish to burden the reader with every equation in the bounded variable format, many equations influenced by the change in form have been omitted. However, we have considered all of the different ways bounded variables affect the mathematics of the thesis. The reader can use the results of this chapter to derive the remaining relations.

REFERENCES

[1] Adler, I., Resende, M.G. and Veiga, G. (1986). An implementation of Karmarkar's algorithm for linear programming, Technical Report ORC 86-8, Operations Research Center, Department of Industrial Engineering and Operations Research, University of California, Berkeley.

[2] Ashford, R. (1986). Devex pricing in the simplex algorithm, Warwick Papers in Management No. 5, Institute for Management Research and Development, University of Warwick, Coventry.

[3] Blair, C. (1982). Some linear programs requiring many pivots, Faculty Working Paper No. 867, College of Commerce and Business Administration, University of Illinois at Champaign-Urbana.

[4] Chang, S.Y. and Murty, K.G. (1987). The steepest descent gravitational method for linear programming, Technical Report #87-14, Department of Industrial and Operations Engineering, the University of Michigan, Ann Arbor, Michigan.

[5] Cooper, L. and Kennington, J. (1979). Nonextreme point solution strategies for linear programs, *Naval Research Logistics Quarterly* **26**, pp. 447–462.

[6] Dantzig, G.B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.

[7] Dantzig, G.B. (1988). Making progress during a stall in the simplex algorithm, Technical Report SOL 88-5, Systems Optimization Lab, Department of Operations Research, Stanford University, Stanford, California.

[8] Eiselt, H.A. and Sandblom, C.-L. (1985). External pivoting in the simplex algorithm, *Statistica Neerlandica* **39**, pp. 327–341.

[9] Eiselt, H.A., Sandblom, C.-L. and DeMarr, R. (1985). Computational experience with external pivoting, Mathematical Programming Society Committee on Algorithms Newsletter No. 12, pp. 16–20.

[10] Eiselt, H.A. and Sandblom, C.-L. (1986). On estimating optimal bases for linear programs, *Journal of Information and Optimization Sciences* **7**, pp. 29–39.

[11] Falkner, J.C. and Ryan, D.M. (1987). Aspects of bus crew scheduling using a set partitioning model, Department of Theoretical and Applied Mechanics, University of Auckland, Auckland, New Zealand.

[12] Fathi, Y. and Murty, K.G. (1986). Computational behavior of a feasible direction method for linear programming, IE Technical Report #86-11, North Carolina State University, Raleigh, North Carolina.

[13] Gass, S. I. and Saaty, T.L. (1955). The computational algorithm for the parametric objective function, *Naval. Res. Logist. Quart.* **2**, pp. 39–45.

[14] Gill, P.E., Murray, W. and Wright, M.H. (1981). *Practical Optimization*, Academic Press, London.

[15] Gill, P.E., Murray, W., Saunders, M.A., Tomlin, J.A. and Wright, M.H. (1986). On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method, *Mathematical Programming* **36**, pp. 183–209.

[16] Gill, P.E., Murray, W., Saunders, M.A. and Wright, M.H. (1987). Maintaining LU factors of a general sparse matrix, *Linear Algebra and its Applications* **88/89**, pp. 239–270.

[17] Goldfarb, D. and Sit, W.Y. (1979). Worst case behavior of the steepest edge simplex method, *Discrete Appl. Math.* **1**, pp. 277–285.

[18] Harris, P.M.J. (1973). Pivot selection methods of the Devex LP code, *Mathematical Programming Study* **4**, pp. 30–57.

[19] Jeroslow, R.G. (1973). The simplex algorithm with the pivot rule of maximizing criterion improvement, *Discrete Mathematics* **4**, pp. 367–377.

[20] Kalan, J.E. (1976). Machine-inspired enhancements of the simplex algorithm, Technical Report CS75001-R, Computer Science Department, Virginia Polytechnical University, Blacksburg, Virginia.

[21] Kallio, M. and Porteus, E. (1978). A class of methods for linear programming, *Mathematical Programming* **14**, pp. 161–169.

[22] Kallio, M. and Orchard-Hays, W. (1981). Experiments with the reduced gradient method for general and dynamic linear programming, in Large Scale Linear Programming (G.B. Dantzig, M.A.H. Dempster and M. Kallio, eds.), pp. 631–662, IIASA, Laxenburg, Austria.

[23] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming, Proceedings of the 16th Annual ACM Symposium on the Theory of Computing, pp. 302–311.

[24] Klee, V. (1965). A class of linear programming problems requiring a large number of iterations, *Numer. Math.* **7**, pp. 313–321.

[25] Klee, V. and Minty, G.J. (1972). How good is the simplex algorithm?, in Proceedings of the 3rd Symposium on Inequalities (Shish, O., ed.), pp. 159–175, Academic Press, New York.

[26] Lustig, I.J. (1985). A practical approach to Karmarkar's algorithm, Technical Report SOL 85-5, Systems Optimization Lab, Department of Operations Research, Stanford University, Stanford, California.

[27] Lustig, I.J. (1987). Comparisons of composite simplex algorithms, Technical Report SOL 87-8, Systems Optimization Lab, Department of Operations Research, Stanford University, Stanford, California.

[28] Lustig, I.J. (1987). The equivalence of Dantzig's self-dual parametric algorithm for linear programs to Lemke's algorithm for linear complementarity problems applied to linear programs, Technical Report SOL 87-4, Systems Optimization Lab, Department of Operations Research, Stanford University, Stanford, California.

[29] Lustig, I.J. (1987). An analysis of an available set of linear programming test problems, Technical Report SOL 87-11, Systems Optimization Lab, Department of Operations Research, Stanford University, Stanford, California.

[30] Megiddo, N. (1983). Linear-time algorithms for linear programming in R^3 and related problems, *Siam J. Computing* **12**, pp. 759–776.

[31] Murtagh, B.A. (1981). *Advanced Linear Programming*, McGraw-Hill, New York.

[32] Murtagh, B.A. and Saunders, M.A. (1978). Large-scale linearly constrained optimization, *Mathematical Programming* **14**, pp. 41–72.

[33] Murtagh, B.A. and Saunders, M.A. (1983). MINOS 5.0 User's Guide, Technical Report SOL 83-20, Systems Optimization Lab, Department of Operations Research, Stanford University, Stanford, California.

[34] Murtagh, B.A. and Saunders, M.A. (1987). MINOS 5.1 User's Guide, Technical Report SOL 83-20R, Systems Optimization Lab, Department of Operations Research, Stanford University, Stanford, California.

[35] Murty, K.G. (1976). *Linear and Combinatorial Programming*, John Wiley and Sons, New York.

[36] Murty, K.G. and Fathi, Y. (1984). A feasible direction method for linear programming, *Operations Research* **3**, pp. 121–127.

[37] Nazareth, L. (1985). Pricing criteria in the simplex method, Computational Decision Support Systems, Berkeley, California.

[38] Perold, A.F. (1981). Exploiting degeneracy in the simplex method, in Large Scale Linear Programming (G.B. Dantzig, M.A.H. Dempster and M. Kallio, eds.), pp. 55–66, IIASA, Laxenburg, Austria.

[39] Perold, A.F. (1981) A degeneracy exploiting LU factorization for the simplex method, in Large Scale Linear Programming (G.B. Dantzig, M.A.H. Dempster and M.J. Kallio, eds.), pp. 67–96, IIASA, Laxenburg, Austria.

[40] Pyle, L.D. (1987). Generalizations of the simplex algorithm, Department of Computer Science, Houston, Texas.

[41] Ravindran, A. (1973). A comparison of the primal simplex and complementary pivot methods for linear programming, *Naval Research Logistics Quarterly* **20**, pp. 95–100.

[42] Renegar, J. (1986). A polynomial-time algorithm, based on Newton's method, for linear programming, Mathematical Science Research Institute, Berkeley, California.

[43] Sethi, A.P. and Thompson, G.L. (1984). The pivot and probe algorithm for solving a linear program, *Mathematical Programming* **29**, pp. 219–233.

[44] Shamir, R. (1987). The efficiency of the simplex method: a survey, *Management Science* **33**, pp. 301–334.

[45] Sherali, H.D., Soyster, A.L. and Baines, S.G. (1983). Nonadjacent extreme point methods for solving linear programs, *Naval Research Logistics Quarterly* **30**, pp. 145–161.

[46] Strang, G. (1976). *Linear Algebra and its Applications*, Academic Press, London and New York.

[47] Todd, M. (1988). Karmarkar as Dantzig-Wolfe, Technical Report #782, College of Engineering, Cornell University, Ithaca, New York.

[48] Todd, M. (1988). Private correspondence.

[49] Vanderbei, R.J., Meketon, M.S. and Freedman, B.A. (1986). A modification of Karmarkar's linear programming algorithm, *Algorithmica* **1**, pp. 395–408.

[50] Wolfe, P. (1962). The reduced gradient method, unpublished manuscript, RAND Corporation.

[51] Wolfe, P. (1963). A technique for resolving degeneracy in linear programming, *J. Soc. Indust. Appl. Math.* **11**, pp. 205–211.

[52] Ye, Y. (1987). Eliminating columns in the simplex method for linear programming, Department of Engineering-Economic Systems, Stanford University, Stanford, California.

[53] Zadeh, N. (1980). What is the worst-case behavior of the simplex algorithm?, Technical Report No. 27, Department of Operations Research, Stanford University, Stanford, California.

[54] Zikan, K. and Cottle, R.W. (1987). The box method for linear programming: part I - basic theory, Technical Report SOL 87-6, Systems Optimization Lab, Department of Operations Research, Stanford University, Stanford, California.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report SOL 88-15	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Dynamic Pricing Criteria in Linear Programming		5. TYPE OF REPORT & PERIOD COVERED **Technical Report** 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Edward S. Klotz		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0343
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL Stanford University Stanford, CA 94305-4022		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1111MA
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Dept. of the Navy 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE July 1988 13. NUMBER OF PAGES 84 Pages
		14. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Linear Programming, Simplex Method, Reduced-Gradient Method, Pricing, Degeneracy.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Please see other side for Abstract...

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

In recent years the interest in linear programming algorithms has increased greatly due to the discovery of new interior-point methods. New results have also prompted researchers to reconsider some previously discarded ideas in light of their additional knowledge. This thesis begins with a study of some variants of the reduced-gradient method applied to linear programs. Preliminary computational tests revealed how sparsity and degeneracy, two characteristics present in most practical problems, can severely inhibit such variants. The development of dynamic pricing criteria to exclude certain columns from the search direction provides a computationally efficient way to alleviate these difficulties. Application to the simplex method yields a pivot rule designed to avoid degenerate pivots. Generalization of the rule yields a cheap method to estimate the step lengths associated with potential incoming nonbasic variables. The result is a set of pivot rules that appear particularly useful on highly degenerate and poorly scaled linear programs. Extensive computational tests are presented.

END

DATE

FILMED

DTIC

11-88